“The Missing Manual系列绝对是入门指南最为明智的选择……”

—— Kevin Kelly，WIRED联合创始人

TURING 图灵程序设计丛书

HTML5秘籍

the missing manual®

The book that should have been in the box®

第2版



[美] Matthew MacDonald 著

李松峰 朱巍 刘帅 译

O'REILLY®

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

译者介绍

李松峰

2006年起投身翻译，出版过译著20余部，包括《JavaScript高级程序设计》《简约至上》等畅销书。2008年进入出版业，目前在图灵公司从事技术图书策划、编辑和审稿工作，并担任《Web+DB Press中文版》杂志编委。

2007年创立知识分享网站“为之漫笔”（cn-cuckoo.com），翻译了大量国外经典技术文章。他经常参加技术社区活动，曾在W3ctech 2012 Mobile上分享“Dive into Responsive Web Design”。2013年1月应邀在金山网络分享“响应式Web设计”，2013年3月应邀在奇虎360分享“JS的国”。

朱巍

图灵公司老编辑，爱读书，乐于结交爱读书之人，乐于传播好书给更多读者。

刘帅

小米软件工程师，毕业于哈尔滨工程大学，获得计算机科学与技术专业学士学位。毕业以来一直从事前端开发工作，先后就职于新浪、阿里巴巴、腾讯、百度、小米，曾参与开发QQBrowser 6、QQ for Windows 8、百度打车WebApp、百度地图等产品。

TURING 图灵程序设计丛书

HTML5秘籍（第2版）

the missing manual®

The book that should have been in the box®

[美] Matthew MacDonald 著
李松峰 朱巍 刘帅 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北 京

图书在版编目（CIP）数据

HTML5秘籍 / （美）麦克唐纳（MacDonald,M.）著；李松峰，朱巍，刘帅译. -- 2版. -- 北京：人民邮电出版社，2015.4
（图灵程序设计丛书）
ISBN 978-7-115-32050-6

Ⅰ. ①H… Ⅱ. ①麦… ②李… ③朱… ④刘… Ⅲ. ①超文本标记语言—程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2015)第036348号

内 容 提 要

本书共包括四个部分，共 13 章。第一部分介绍了 HTML5 的发展历程，用语义元素构造网页，编写更有意义的标记，以及构建更好的 Web 表单。第二部分介绍了 HTML5 中的音频与视频、CSS3、Canvas 绘图技术等内容。第三部分介绍了数据存储、离线应用、与 Web 服务器通信，以及 HTML5 与 JavaScript 技术的强大结合等内容。第四部分为附录，简单介绍了 CSS 和 JavaScript。

本书既适合新手学习，也能助有经验的 Web 开发人员解决日常工作中遇到的难题。

◆ 著　[美] Matthew MacDonald
译　李松峰　朱　巍　刘　帅
责任编辑　岳新欣
责任印制　杨林杰

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：26.25
字数：620千字　2015年 4 月第 2 版
印数：1 – 3 000册　2015年 4 月北京第 1 次印刷
著作权合同登记号　图字：01-2014-6468号

定价：89.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

版权声明

O'Reilly Media, Inc.介绍

O'Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O'Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项 O'Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar 博客有口皆碑。”

——*Wired*

“O'Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O'Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O'Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去 Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

目　　录

前　　言

乍一看，你可能觉得HTML5是网页编写语言HTML的第5个版本。但实际上，这背后的故事可乱得多。

HTML5是一个叛逆。它是由一群自由思想者组成的团队设计出来的，这个团队的成员并不负责制定官方HTML标准。它允许使用10年前就被禁止的网页编写方式。它费尽心机、苦口婆心地告诉浏览器开发商怎么处理而不是彻底拒绝标记中的错误。它最终实现了不依赖Flash等浏览器插件播放视频。而且它引入了一大批JavaScript驱动的功能，让网页可以像传统桌面软件那样丰富多彩、富有交互能力。

理解HTML5可没有那么简单。最主要的困难在于人们用HTML5这个词指代十几种甚至更多独立的标准。（后面我们会介绍到，这是HTML5发展演进的结果。一开始时它只有一个标准，但后来就拆分成了很多容易管理的分支。）事实上，HTML5现在代表的是“HTML5及所有相关标准”，甚至还可以更宽泛，代表“下一代网页编写技术”。这就是本书要带领大家探索的HTML5：既包括HTML5核心语言，也包括与HTML5纠缠在一块但在其标准中永远找不到的那些新功能。

于是，第二个困难又摆在了你的面前：浏览器支持。不同的浏览器支持HTML5的程度不同。最差劲的是IE8，它对HTML5的支持非常有限，目前每20台能上网的电脑中就有1台安装着IE8（至少本书写作时是这个比例，后面1.6.2节会介绍怎么查到浏览器装机统计信息）。好在我们有办法弥补浏览器支持上的缺陷，但有的办法简单，有的办法也不讨人喜欢。根据在今天的网页中使用HTML5的需求，本书对这两种方法都会适当介绍。

抛开这些困难，有一个事实毫无争议：HTML5代表未来。苹果、谷歌和微软等大软件公司都在鼎力支持它；W3C（World Wide Web Consortium，万维网联盟）已经放弃了XHTML，从而使HTML5成为正式标准并得到认可。如果你在看这本书，那么也可能加入HTML5阵营，并利用它创造出如图0-1所示的那种酷炫的网页。

阅读本书的条件

本书介绍的HTML5是HTML标准最新最好的版本。虽然不一定非得是标记大师才能看懂这本书，但阅读本书的的确确还是需要一些Web设计经验的。以下就是几个必要条件。

- **写过网页**。本书假设你以前至少写过一些网页（或者至少知道怎么使用HTML元素把内容分成标题、段落和列表，等等）。如果你刚刚接触Web设计，那最好是先找一本合适的入

门书看一看，比如我的*Creating a Website: The Missing Manual, Third Edition*。（不过别担心，你不会被限制在过去的技术中，这本书里的示例都是有效的HTML5文档。）

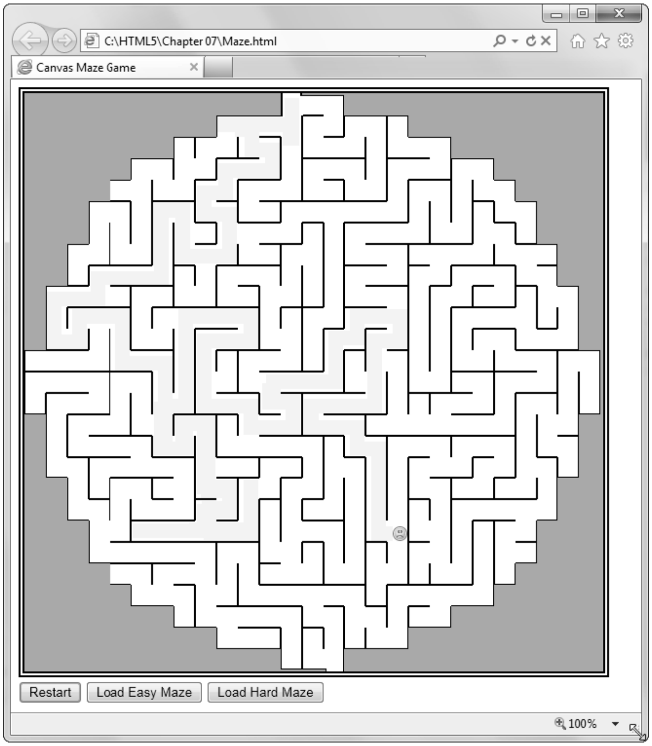


图0-1：在Web世界黑暗的过去，要编写网页游戏，必须依赖Flash这样的浏览器插件。但有了HTML5的新功能——包括canvas（图中显示的就是）之后，你可以使用可靠而又免插件的JavaScript。这个图展示的是用HTML5技术开发的迷宫游戏（第9章将详细讨论）

- **懂样式表**。没有CSS（Cascading Style Sheet，层叠样式表）就没如今的网站。CSS为页面提供布局和格式。要想顺利阅读本书，你应该知道样式表的基本知识，包括怎么创建样式表，里面都有什么，以及怎么把它应用到网页上。如果你不太清楚CSS是干什么的，可以先看一看附录A。如果你需要更多帮助，或者想提高自己的CSS技能，以便真正做出漂亮的布局和样式，建议你看看David Sawyer McFarland的*CSS3: The Missing Manual*（O'Reilly）。
- **懂JavaScript**。当然，编写HTML5页面用不着JavaScript。可是，如果你想使用HTML5不计其数的强大功能——比如在画布上画图或者与Web服务器通信，那就需要JavaScript了。如果你有一些浅显的编程经验，但对JavaScript还一知半解，附录B可以帮你掌握一些新情况。不过，要是一听到写代码这几个字，马上就像被窝里爬进一条蟒蛇那样魂飞魄散，那要么你根本不必看本书中的很多章节了，要么你得通过David Sawyer McFarland的*JavaScript & jQuery: The Missing Manual*（O'Reilly）补补课。

如果这些必要条件让你头晕目眩——好吧，这就是活在Web设计最前沿必须付出的代价。

编写 HTML5

编写HTML5页面可以使用编写HTML页面时使用的软件。可以是个再简单不过的文本编辑器，像Windows中的记事本，或者Mac中的TextEdit。目前也有很多设计工具（比如Adobe Dreamweaver和Microsoft Visual Studio）提供了快速创建新HTML5文档的模板。不过，HTML5页面的基本结构确实非常简单，任何网页编辑软件（即使不是为HTML5设计的）都没有问题。

注意 当然啦，不管你上网和编写网页时用的计算机是Windows PC，还是最新的MacBook Pro，同样也无所谓，因为HTML5与操作系统无关。

查看 HTML5

现在，任何浏览器的最新版本都能支持大多数的HTML5功能，包括那些在苹果和Android设备上运行的移动版浏览器。只要你的浏览器是最新的，HTML5就可以出色地运行，因此就可以用来测试本书中的示例。

目前，还没有一款浏览器巨细靡遗地支持HTML5的所有功能，HTML5本身就是一大批互相关联的标准也是其中一个原因。谷歌Chrome在对HTML5的支持方面通常处于领跑地位，Firefox和Opera紧追不舍，Safari也尾随其后，IE的差距则一直很大。老版本IE是真正的老大难问题，这些IE不能升级，主要是因为它们都运行在Windows Vista或Windows XP这样老掉牙的操作系统上（这两个操作系统仍然跑在世界1/5的台式机中）。1.6节会更全面地探讨这个问题，并给出应对建议。

什么时候可以使用 HTML5

简短的答案是“现在”。就连遭人唾弃的IE6，这个问世长达10年之久、补丁摞补丁的家伙都可以显示基本的HTML5文档。这是因为创建HTML5标准时，就想让它能涵盖并扩展原来的HTML。

更详尽的答案是“视情况而定”。前面刚刚提到过，HTML5是一组不同标准的集合，浏览器对这些标准有着不同程度的支持。因此，尽管现在任何Web开发人员都可以转而编写HTML5文档（谷歌、YouTube和Wikipedia等很多大型网站已经这样做了），但要放心地使用HTML5的全部新奇功能——至少不必针对那些不够开化的浏览器采取变通手段，恐怕还要再过一些时日。

注意 在告诉读者使用某项HTML5新功能之前，我们会明确指出该功能得到了哪些浏览器的支持。当然，浏览器版本变化得比较快，如果你担心使用某功能会有问题，最好还是随时查阅某些资料。在此，推荐大家使用http://caniuse.com，通过这个网站可以查到任何一项功能，以及哪个浏览器的哪个版本支持它。（1.6.1节还将详细介绍如何使用这个网站。）

作为有标准意识的开发人员，恐怕你也对这些标准什么时候正式颁布感兴趣。但这个问题有点复杂，因为设计HTML5的人遵循的理念有点不合常规。他们经常说，正式的标准中怎么说并不重要，关键是有没有浏览器支持。换句话说，只要你觉得可行，现在就可以采用任何你想使用的功能。但不少开发人员、大公司、政府机关以及其他组织，通常会根据一种语言的标准是否正式颁布来判断是否可以采用它。

在本书写作时，HTML5正处于候选推荐标准（candidate recommendation）阶段。这意味着标准很大程度上已经尘埃落定，但浏览器开发商仍然在完善自己的实现。而进入下一个（也是最后一个）阶段，也就是推荐标准（recommendation）阶段，可能要等到2014年年底。在此期间，W3C已经发布了HTML5.1的工作草案（working draft）。（要了解这两个版本的区别，请看下面的附注）。

HTML5与HTML5.1的区别

HTML又有新版本了？怎么又多出来一个空格呢？

正如我们将在第1章介绍的，HTML5经过了两个组织之手。整个过程因此遗留了一些小问题，包括一个有点轻微“分裂”的版本系统。

最初制定HTML5规范的人（即WHATWG成员，将在1.1.3节介绍）对版本号不太“感冒”，他们认为HTML5是一门“活的”语言，鼓励Web开发者关注浏览器的支持情况，而非版本号。

可是，WHATWG把HTML5交给了W3C官方，以便W3C把它制定完成。W3C是一个严谨、细致的组织，希望区分最初发布的HTML5和后来经过改进的版本。因此，W3C决定将最初发布的HTML5命名为HTML 5.0（注意空格），第二次发布的版本就是HTML 5.1，第三次发布时就叫HTML 5.2。真够乱的，所有这些版本其实都还是HTML5。

顺便说一下，后续几个版本的HTML5标准不可能再有大的改动了。如果有新功能，也会以独立的、补充性的规范形式发布。这样，少数几个人很快就可以制定新的HTML5功能规范，而不必等待对整个语言的修订。

本书内容

本书把完整的HTML5教程分为13章，具体内容如下。

第一部分：现代标记

- 第1章介绍HTML发展到HTML5的历程。我们会看一看HTML5文档的样子，看看它跟以前的HTML有何不同，另外也看一下浏览器的支持情况。
- 第2章讨论HTML5的语义元素（semantic element），也就是一组可以为标记赋予含义的元素。恰当地使用这些元素，可以让浏览器、屏幕阅读器、Web设计工具以及搜索引擎基于它们提供的额外信息更智能地工作。
- 第3章进一步讨论语义的概念，涉及微数据（microdata）等标准。尽管这一章的内容有点偏理论，但透彻理解这个概念可以给Web开发人员带来巨大的回报：在谷歌等搜索引擎的

结果列表中显示更全面、更详尽的内容。

- 第4章探索HTML5 Web表单元素的变化，包括文本框、选择列表、复选框和其他用来从访客那里收集信息的微件（widget）。HTML5为捕获数据输入错误提供了一些辅助和基本工具。

第二部分：视频、图形和特效

- 第5章讲一讲HTML最激动人心的功能，即支持音频和视频播放。这一章将介绍如何避免遭遇“Web视频编解码器大战”，创建出在所有浏览器中都能工作的播放页面，同时还要学习创建自己定制的播放器。
- 第6章介绍最新版本的CSS3标准，它与HTML5可谓绝配。我们将学习如何应用新奇的字体让文本变得活泼可爱，以及如何利用变换和动画添加吸引人的效果。
- 第7章探索CSS3媒体查询。我们会介绍怎么使用它创建响应式设计，让网页平滑地适配各种移动设备。
- 第8章将介绍名为画布（canvas）的二维绘图表面。你将会学习怎样在画布上绘制图形、图像、文本，甚至还将构建一个简单的绘图程序（使用JavaScript）。
- 第9章进一步提升你的“绘画”技术。这一章将会学习投影、花哨的模式，以及可点击的交互图形和动画等更加令人神往的Canvas技术。

第三部分：构建 Web 应用

- 第10章讨论在访客计算机中保存小段数据的Web存储功能。这一章还将介绍如何在网页而不是在Web服务器中使用JavaScript代码处理用户选择的文件。
- 第11章探索HTML5的缓存功能，这个功能可以实现在断网的情况下仍然能够通过浏览器查看网页。
- 第12章将把目光投向与Web服务器通信这个主题上。为此，将介绍久负盛名的`XMLHttpRequest`对象，JavaScript通过它可以联系Web服务器并请求信息。然后再讨论两个比较新的功能：服务器发送事件（Server-Sent Events）和Web Socket。
- 第13章介绍了解决现代Web应用开发难题的三个新功能。第一是可以确定访客位置的地理定位；第二是在后台执行复杂任务的Web Worker；第三是能够同步网页URL到当前状态的浏览器历史功能。

最后有两个附录，可以为你掌握HTML5补习一些基础知识。附录A是对CSS的一个简要介绍，附录B则会简单地介绍JavaScript。

在线资源

作为Missing Manual丛书的读者，你所得到的不仅仅是一本书。在网上，你还可以找到示例文件以及技巧、文章，甚至是一两段视频。你可以跟Missing Manual团队交流，告诉我们你喜欢

（或讨厌）本书的哪一方面。请访问www.missingmanuals.com，或直接阅读后面的某一小节。

Missing CD

本书没有附带光盘，但这对学习本书一点影响都没有。读者可以访问本书的Missing CD页面http://missingmanuals.com/cds/html5tmm2e，下载本书讨论和展示的网页示例，这样你就不必自己动手敲那些长长的网页地址了。这个页面中列出了全书每一章提到的网站的链接。

提示 假如你想找某个特定的例子，我教给你一个好办法——看插图。在插图中，文件名一般都会出现在浏览器地址栏的末尾。比如，看到文件路径C:\HTML5\Chapter01\SuperSimpleHTML5.html（图1-1），就知道对应的示例文件名叫SuperSimpleHTML5.html。

试验站点

还有另一种使用本书示例的方法，就是访问在线示例网站：www.prosetech.com/html5。在这个网站上可以看到本书的每一个示例，并直接在浏览器中运行它们。因为HTML5的某些功能需要一个真正的Web服务器，所以直接使用这个网站其实可以省点心。（如果你直接从计算机硬盘上运行网页，这些功能可能会导致一些怪异的现象，或者完全不能用。）而使用这个网站，就可以先看到某个例子的运行结果，然后再下载该页面并动手尝试。

注意 别担心自己不知道哪些HTML5功能需要Web服务器，到时候本书会给出提示的。

注册

如果你在oreilly.com注册了这本书，可能会享受到一些优惠，比如购买*HTML5*: *The Missing Manual*的新版时可以打个折。注册其实只需点几次鼠标。在浏览器地址栏里输入http://tinyurl.com/registerbook，直接就可以跳到注册（Registration）页面。

反馈

有问题要问？需要更多信息？想给我们写个书评？在反馈（Feedback）页面上，你可以向专家请教自己看书时碰到的问题，也可以分享自己对Missing Manual丛书的看法，甚至找到一些志同道合的朋友，听听他们谈论在做网站过程中的一些体会。

要想发言，可访问www.missingmanuals.com/feedback。

勘误

为了尽可能保证本书切合实际、准确无误，每次重印我们都会纠正一些确认的勘误。这些勘

误信息也会在本书网站上发布出来，以便读者更正自己手里这本书的错误。要提交或查看勘误，请访问http://tinyurl.com/html52e-mm[①]。

Safari® Books Online

Safari® Books Online是一个按需阅读的数字图书馆，有7500种技术图书和视频可供搜索。

通过订阅，可以在此阅读所有图书，观看任何视频。甚至可以在新书印刷之前阅读到它们。可以复制粘贴示例代码，收藏喜欢的内容，下载整章内容，为关键部分创建书签，添加评注，打印页面，以及享受其他众多省时省力的阅读体验。

O’Reilly Media已经将本书上传到Safari Books Online。访问http://my.safaribooksonline.com并免费注册，可以看到本书及O’Reilly和其他出版社图书的完整电子版。

① 要提交中文版勘误，请访问图灵社区，并在本书页面下提交：http://ituring.cn/book/1361。——编者注

Part 1

第一部分

现代标记

本部分内容

第1章

HTML5简介

如果说HTML是一部电影，那HTML5就是一次大转折。HTML本来是不会活过21世纪的。官方Web标准组织W3C早在1998年就已经对HTML撒手不管了。W3C把未来都寄托在XHTML这个更具现代特色的后续标准身上，XHTML被视为HTML的严肃整洁版。但XHTML举步维艰，是一群被剥夺了话语权的人，让HTML起死回生并为本书将要探讨的功能奠定了基础。

在本章里，你会了解HTML死亡的原因，以及它又是怎样复活的；了解HTML5的设计原理与功能；还将见识恼人的浏览器支持问题。在这一章，你将第一次看到真正的HTML5文档。

1.1　HTML5 的故事

HTML的基本思想——使用*元素*为内容添加结构——从Web诞生以来就没有变过。事实上，即使是最陈旧的网页，在最新的浏览器中仍然可以得到完美的呈现。

年长和成功也会带来风险，那就是所有人都想取代你！1998年，W3C停止了对HTML的维护，作为对它的改进，开始制定一个基于XML的后续版本——XHTML 1.0。

1.1.1　XHTML 1.0：更严格的标准

XHTML与HTML的语法绝大部分都是相同的，只不过要求更严格。很多以前不够严谨的HTML标记，在XHTML中都变成了不能接受的。

例如，假设你想把标题中的最后一个词标记为斜体，本来应该写：

```
<h1>The Life of a <i>Duck</i></h1>
```

但你一不小心放错了最后两个标签的位置：

```
<h1>The Life of a <i>Duck</h1></i>
```

浏览器在遇到这个稍微有点乱的标记之后，能明白你想干什么。于是，它就一声不吭地把最后一个词变成斜体。可是，标签不匹配违反了XHTML的规定。如果把页面复制到一个XHTML验证器中（或使用Dreamweaver之类的网页设计工具时），你就会看到一个警告，告诉你哪里有错误。从Web设计的角度看，XHTML基于严格规则的这种提示很有用，因为你可以发现微小的错误，这些错误会导致在不同浏览器中显示结果不一致（这些错误在编辑和增强页面时还可能导致更严重的问题）。

最初，XHTML获得了成功。由于厌倦了浏览器的古怪行为和怎么写都可以通过的不正常状态，专业的Web开发人员对XHTML还是非常拥护的。后来，XHTML标准又强迫他们养成更好的习惯，同时放弃HTML中那些并不完善的格式化功能。可是，与XML工具协同，降低自动化程序处理页面的难度，方便地移植到移动平台，以及XHTML语言自身的可扩展性等这些预期的好处，从来没有在XHTML身上实现过。

即便如此，XHTML仍然成为最严肃的Web设计师所遵循的标准。尽管看起来所有人都挺满意的，但实际上却存在一个潜规则：浏览器虽然理解XHTML标记，但却不会严格地按照标准执行错误检查。这就意味着页面仍然可以不遵守XHTML规则，浏览器则视而不见。事实上，没有什么可以阻止Web开发人员把乱糟糟的标记和陈旧的HTML内容混在一起，然后还说这是XHTML页面。世界上根本就没有一个浏览器站出来反对这种行为。这种情况让那些负责XHTML标准的人深感不安。

1.1.2　XHTML 2：意想不到的失败

解决方案就是通过XHTML 2来扭转这个乱糟糟的局面。这个新版本规定了严格的错误处理规则，强制要求浏览器拒绝无效的XHTML 2页面，同时也摒弃了很多从HTML沿袭下来的怪异行为和编码惯例。比如，以编号方式（`<h1>`、`<h2>`、`<h3>`等）区分标题的方法被一个新的`<h>`元素取代，这个元素的重要性取决于它在网页中的位置。类似地，由于允许Web开发人员将任何元素转换为链接，`<a>`元素的地位一落千丈。而`<img>`元素因为增加了一种提供替代内容的新方式，也丧失了原有的`alt`属性。

这些变化是XHTML 2的典型特征。从理论上看，这些改变的目的是让网页更整洁也更有逻辑性。而从实践角度说，这就要求Web设计人员必须改变以前编写网页的方式（已经存在的网页必须更新），但付出这些代价却没有增加任何新功能，让这种改变失去了价值。与此同时，XHTML 2还宣布作废了几个众所周知的元素，比如用于加粗文本的`<b>`、用于变斜体的`<i>`和用于在网页中嵌入另一个网页的`<iframe>`，但这些元素在Web设计人员中仍然深得人心。

但最糟糕的，还是慢得要死的制定过程。XHTML 2的制定过程整整拖了5年才完成，开发人员的激情早已荡然无存了。

1.1.3　HTML5：起死回生

几乎与此同时，从2004年开始就有一群人从另外一个角度展望Web的未来。他们想的不是从HTML中挑出各式各样的毛病（或者干脆说是主张“不纯粹的哲学观”），而是它还缺少什么Web开发人员编码时急需的功能。

归根结底，HTML最早是作为显示文档的手段出现的。辅之以JavaScript，它其实已经演变成了一个系统，可以开发搜索引擎、网上商店、在线地图、邮件客户端以及其他各种能够想象得到的Web应用。虽然设计巧妙的Web应用可以实现很多令人赞叹的功能，但开发这样的应用远非易事。多数Web应用都得手动编写大量JavaScript代码，还要用到一个或多个流行的JavaScript工具包，乃至

在Web服务器上运行的服务器端模块。要让所有这些方面在不同的浏览器中都能紧密配合不出差错是一个挑战。即使是赢得了挑战，你还要记住把这些方面联系到一起的那些错综复杂的细节。

开发浏览器的人对这种情况特别关注。于是，来自Opera Software（开发Opera浏览器的公司）和Mozilla Foundation（开发Firefox浏览器的组织）的一些具有超前意识的人纷纷建言，希望XHTML能加入一些对开发人员更有用的功能。但他们的建议并没有被采纳，结果Opera、Mozilla和苹果公司自发地组建了WHATWG（Web Hypertext Application Technology Working Group，Web超文本应用技术工作组），致力于寻找新的解决方案。

WHATWG的目的不是抛弃HTML，而是考虑以无障碍、向后兼容的方式去扩展它。这个组织最早的工作成果包含两个补充规范：Web Application 1.0和Web Forms 2.0。而HTML5正是在这两个标准的基础上发展起来的。

注意 HTML5中的数字5表示这个标准是HTML的后续版本（在XHTML之前，HTML的版本号是4.01）。当然，这个解释也不完全正确，因为HTML5支持自HTML 4.01发布以来的10年间出现在网页中的所有新东西，包括严格的XHTML风格的语法（只要你愿意就可以用）和大量的JavaScript创新。但不管怎么说，这个名字仍然清楚地表明：HTML5虽然支持XHTML的**规定**，但它要求的则是HTML的**规则**。

2007年，WHATWG获得了全球Web开发人员空前的支持。痛定思痛之后，W3C宣布解散负责制定XHTML 2标准的工作组，并开始致力于将HTML5改造为正式的标准。就这样，最初的HTML5被分割成多个容易管理的模块，而本来统称为HTML5的很多功能分散到了几个独立的标准中（更多信息，请详见后面的附注栏）。

提示 W3C官方HTML5标准的网址是www.w3.org/TR/html5。

HTML5包含哪些功能

HTML5包含多个标准，这些标准之间彼此关联。这种局面既好又不好。说好，是因为浏览器可以迅速实现HTML5中业已成熟的部分，而任由其他部分继续发展。说不好，则是因为编写网页的人必须检查浏览器是否支持自己想用的功能。本书将会介绍一些检测浏览器的技术，有的很麻烦，有的则没有那么麻烦。

以下列出了HTML5涵盖的一些主要功能。

- **HTML5核心**。这一部分主要由W3C官方的规范组成，涉及新的语义元素（第2章和第3章）、新的增强的Web表单微件（第4章）、音频和视频支持（第5章）以及通过JavaScript绘图的Canvas（第8章和第9章）。
- **曾经属于HTML5的功能**。这一部分源自WHATWG最初制定的HTML5规范，其中大多数功能需要JavaScript且支持富Web应用开发。最重要的包括本地数据存储（第10章）、

离线应用（第11章）和消息传递（第12章），但本书要介绍的内容还不止这些。

- **有时候会被称为HTML5的功能**。这些通常是指下一代功能，虽然它们从未进入过HTML5标准，但人们还是经常会把它们与HTML5相提并论。这部分包括CSS3（第6章和第7章）和地理定位（第13章）。

甚至连W3C都在有意无意地模糊“真正的”HTML5（已有标准）和“宣传用”版本（包括HTML5的所有新增部分和补充规范）之间的界限。举个例子，官方W3C标志网站（www.w3.org/html/logo）鼓励人们生成用于宣传CSS3和SVG的HTML5标志，而前两个标准在HTML5出现之前就已经在开发了。

1.1.4　HTML：活着的语言

从W3C到WHATWG，然后再回到W3C，这个过程导致了相当罕见的转换与磨合。从技术上说，什么是或什么不是HTML5由W3C说了算。但与此同时，WHATWG一直在设计未来的HTML功能。直到最近，他们才不再把自己的工作成果称为HTML5，而是简单地称为HTML，表明HTML还会继续活下去。

因为HTML是一门活着的语言，所以HTML页面永远不会作废，也不会无法阅读。HTML页面永远不需要版本号（甚至连文档类型声明都不需要），Web开发人员也永远不需要为了让它能在新浏览器中运行，而把自己的标记从一个版本“升级”到另一个版本。同样，任何时候在HTML标准中都可能增添新功能。

Web开发人员听到这么说，第一反应通常是大惑不解。毕竟，谁希望浏览器对标准的支持各不相同，而谁又愿意在选择功能时只凭它们将来会得到支持这个可能性呢？然而，冷静下来想一想，大多数Web开发人员还是不情愿地接受了这个现实：无论如何，浏览器从Web诞生的那一天起始终都是这样的。

前面我们解释过，今天的浏览器乐于接受支持一大堆乱七八糟的功能这个现实。你可以在激进的XHTML页面中加上像`<marquee>`元素（用于创建滚动文本，已废弃）这样被认为是倒行逆施的东西，任何浏览器都不会反对。类似地，即便是在对最老标准的支持方面，有些浏览器也仍然存在一些广为人知的遗漏。比如，有些浏览器开发商在完整地支持CSS2之前就开始实现CSS3，结果很多CSS2特性后来都没有实现。唯一的区别就是HTML5现在把“活着的语言”变成了常规状态。同样，就像我们正在用新的、创新性的一章来介绍HTML一样，它经过了一番轮回终于又恢复了它的本来面貌，这不也正是一个天大的讽刺吗？

提示　要了解当下正在发展中的HTML，包括被称为HTML5的部分和少量但始终在变化的、新的、还没有得到支持的功能，请访问http://whatwg.org/html。

1.2 HTML5 的三个主要原理

此时此刻，有的读者可能已经按捺不住了，迫不急待地想知道真正的HTML5页面到底是什么样子的。不过在此之前，有必要先了解一下制定HTML5规范的那些人当时是怎么想的。只有理解了这门语言背后的设计思想，才能真正明白本书将要介绍的那些古怪的行为、复杂的现象和偶尔会让人抓耳挠腮的问题。

1.2.1 不破坏Web

“不破坏Web”的意思是标准不应该引入导致已有的网页无法工作的改变。幸运的是，这种事故极少发生。

“不破坏Web”还意味着标准不应该出人意料地更改规则，不能认定今天还完美无缺的网页到了明天就要作废（即使可以使用也要作废）。比如，XHTML 2 破坏了Web，因为它要求马上就显著改变以前编写网页的方式。没错，原来的网页还能用，但那都是浏览器支持向后兼容的功劳。如果你为将来打算，想按照最新标准重写网页，就得浪费数不清的时间去纠正XHTML 2 已经明令禁止的“错误”。

HTML5的立场不一样。HTML5之前可以接受的，在HTML5中照样可以接受。事实也是，符合HTML 4.01标准的网页在HTML5中仍然是有效的。

注意 与以往的标准不同，HTML5不仅向浏览器开发商明示该支持什么，还利用文档说明并规范化了它们**原来的处理方式**。由于HTML5标准描述的都是事实，而不是抛出一堆理想的规则了事，因此它有望成为有史以来受支持程度最高的Web标准。

HTML5怎么处理废弃元素

因为HTML5支持所有HTML，所以它支持很多被认为是废弃的功能。其中包括像`<font>`这样的格式化元素，被人厌恶的`<blink>`和`<marquee>`等特效元素，以及难对付的HTML框架体系。

这种无所不包的开放性是令很多HTML5新手感到困惑的一个原因。一方面，HTML5无论如何还是应该禁止使用这些过时的元素，因为它们已经很多年没有出现在官方规范里了。另一方面，现代浏览器依然悄无声息地支持着这些元素，而HTML5就是要体现浏览器真实的处理方式。那么这个标准到底要怎么做呢？

为解决这个问题，HTML5规范包含两个独立的部分。第一部分（也是本书将要介绍的）面向Web开发人员，要求摒弃过去的那些坏习惯和被废弃的元素。通过使用HTML5验证器可以确保遵循HTML5标准的这一部分。

第二部分，也是HTML5规范中篇幅更长的部分，针对的是浏览器开发商。它们需要支持HTML中存在的一切，以做到向后兼容。理想情况下，HTML5标准中应该包含足够的信息，

让人能够据以从头开发一个新浏览器，而且无论是处理新的还是旧的标记，该浏览器都应该能够与今天的现代浏览器完全兼容。这一部分标准就是告诉浏览器如何处理那些官方不鼓励使用但仍然必须支持的废弃元素。

有时候，HTML5规范也会对浏览器应如何处理各种错误（如漏写或错配了标签）作出正式规定。这一点其实很重要，因为它可以确保有缺陷的页面在不同浏览器中都能够得到同样的处理，甚至都规定了将页面映射为DOM（Document Object Model，文档对象模型，即内存中表现页面元素的对象树，供JavaScript使用）这么细节的问题。为了写出标准的这个冗长又乏味的部分，HTML5的制定者们在现代浏览器上进行了彻底的测试，以便发现还没有作出规定的错误处理行为。然后再把该行为加到标准中。

1.2.2 修补牛蹄子路

牛蹄子路（cowpath）指的是高低不平但使用频率很高的路，通过它可以从一个地方到达另一个地方。之所以存在牛蹄子路，就是因为有人走。也许这条路走起来不是最舒服的，但某种程度上却是最实际的解决之道。

HTML5标准化了这些非官方（但广泛应用）的技术。或许与利用新方法修的高速公路相比，牛蹄子路没有那么平坦宽阔，但它赢得胜利的机会更大。因为对于一般的网站设计人员来说，切换到新技术可能会超出他们的能力范围，或者他们根本就没有兴趣。更大的问题在于，使用旧浏览器的访客无法因为新技术而受益。XHTML 2企图把人们赶出牛蹄子路，结果败得非常惨。

注意 修补牛蹄子路有一个明显的好处：它使用已经得到浏览器某种程度支持的既定技术。假设有一种只被七成浏览器支持的漂亮的新技术，还有一种任何情况都能工作但不那么雅观的hack，Web开发人员始终都会选择不那么雅观的hack，因为它适合更多的用户。

“修补牛蹄子路”的方法也需要折中。有时候，这意味着要包容那些得到广泛支持但设计却很拙劣的功能。HTML5的拖放就是一个例子（参见10.3.5节），这个功能完全以微软为IE5设计的拖放机制为基础。尽管这个拖放功能目前得到了所有浏览器的支持，但由于使用起来不灵活而且过度复杂，因此几乎没有人不反感它。为此，不少的Web设计人员也抱怨“HTML5不仅鼓励不良行为，而且还给它们正名。”

1.2.3 实用至上

这个原理很简单：改变应该以实用为目的。改变得越多，代价也就越大。Web开发人员可能更希望标准是精心设计、始终如一，而且是没有怪异行为的。但这个理由还不足以改变一门已经用来创建了数十亿网页的语言。当然，到底需要不需要改变还是要由某个人根据利害来评判。而现有网页都是怎么做的或者说试图怎么去做，可以作为很好的判断依据。

例如，（在本书写作时）YouTube是世界上第三受欢迎的网站，但由于HTML5之前的HTML不真正支持视频，YouTube一直都得依赖Flash插件。使用Flash插件没什么问题，因为只要是能上网的计算机，基本上都会安装这个插件。不过也有例外，比如某些公司会锁定它们的计算机，不允许安装Flash，另外一些移动设备（如iPhone、iPad和Kindle）也不支持Flash。不管有多少计算机安装了Flash，扩展HTML标准，使其直接支持人们今天使用Web的一种最基本方式——看视频，毋庸置疑是有必要的。

而HTML5中添加了更多交互功能的背后也有着同样的动机。像拖放、可编辑的HTML内容、在Canvas中绘制二维图形等，都是同样的情况。这些功能在我们身边的网页中并不鲜见，只不过有的通过Adobe Flash或微软Silverlight等插件实现，而有的则是利用JavaScript库或（更艰苦地）完全通过手工编写JavaScript代码来实现。因此，为什么不在HTML标准中加入官方的支持，让这些功能在所有浏览器中都能一致地工作呢？HTML5正是要这么做。

注意 Flash等浏览器插件不会一夜之间就消失。尽管HTML5有很多创新，但通过它来构建复杂的图形界面应用，仍然不是件轻而易举的事。不过，HTML5的终极目标很清楚：让网站不依赖插件也能够提供视频、丰富的交互功能以及各种漂亮的效果。

1.3 HTML5 标记初体验

下面是一个最简单的HTML5文档。

```
<!DOCTYPE html>
<title>A Tiny HTML Document</title>
<p>Let's rock the browser, HTML5 style.</p>
```

开始是HTML5的文档类型声明（下一节会详细介绍），然后是页面标题和一些内容。在这里，内容是包含在一个段落中的文本。

想必读者已经知道它在浏览器中是个什么样子了，不过为了验证你的直觉，可以参考图1-1。

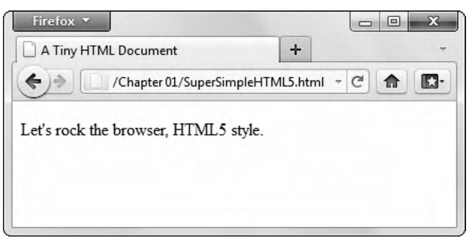


图1-1：只包含一行文本的超简单的HTML5文档

甚至还可以进一步给这个文档瘦身。比如，HTML5标准不要求必须有最后面那个</p>标签，因为浏览器知道在文档后面要关闭所有没有关闭的标签（HTML5标准规定浏览器必须这样处理）。可是，这种简单的写法会让标记显得很乱，甚至可能导致意料之外的错误。

如果有其他方式提供标题信息，HTML5标准也允许你省略<title>元素。比如，在通过电子邮件发送HTML文档时，可以把标题放在邮件的标题中，而把其他标记（文档类型声明以及内容）放在邮件的正文里。不过，这很明显是一种特例。

更常见的情况，则是充实这个已经瘦骨嶙峋的HTML5文档。大多数Web开发人员都认为使用<head>和<body>来分块可以避免导致混乱，因为可以把关于页面的信息（头部）与页面的实际内容（主体）分开。在为页面添加脚本、样式表和元数据的时候，这种结构特别实用：

```
<!DOCTYPE html>
<head>
  <title>A Tiny HTML Document</title>
</head>
<body>
  <p>Let's rock the browser, HTML5 style.</p>
</body>
```

当然，（第3行和第6行）代码中的缩进不是必需的。这里使用缩进就是为了让人一眼就能看清楚页面结构。

最后，还可以选择用<html>元素来封装整个文档（不包含文档类型声明那一行）。结果就成了这样：

```
<!DOCTYPE html>
<html>
<head>
  <title>A Tiny HTML Document</title>
</head>
<body>
  <p>Let's rock the browser, HTML5 style.</p>
</body>
</html>
```

在HTML5以前，所有版本的HTML官方规范都要求使用<html>元素，而实际上用不用它对浏览器来说是无所谓的。HTML5则规定可用可不用。

注意 使不使用<html>、<head>和<body>元素只代表一种风格。即便是在HTML5诞生之前就已经存在的浏览器中，不用这些元素，页面照样可以完美呈现。事实上，浏览器会自动假设页面中已经包含了这些元素。因此，如果用JavaScript来查询DOM（表示页面中元素的一组对象，可以通过编程方式访问），仍然能够找到<html>、<head>和<body>元素，无论你实际上加了还是没加。

现在，这个示例比最简单的HTML5文档复杂一些，但比真正实用的HTML5网页又简单一些。接下来的几节，我们会陆续向其中加入新内容，同时也对标记进行简单介绍。

1.3.1 HTML5 文档类型

每个HTML5文档的第一行都必须是一个特定的文档类型声明。这个文档类型声明用于宣告

后面的文档标记遵循哪个标准。以下文档声明表示文档标记遵循HTML5标准：

```
<!DOCTYPE html>
```

HTML5的文档类型声明给人的第一印象就是极其简单。特别是与冗长的XHTML 1.0严格型的文档类型声明相比，这一点更明显：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

面对如此复杂的XHTML文档类型声明，就连专业的Web开发人员也不得不采用复制粘贴的方法。相比之下，HTML5的文档类型声明简约至上，手工输入也不麻烦。

另外，HTML5的文档声明还有一点值得注意，那就是它不包含官方规范的版本号（即HTML5中的5）。事实上，这个声明仅仅表明当前页面是HTML页面。这与HTML5作为一门活着的语言（见1.1.4节）的远见是分不开的。换句话说，只要有新功能添加到HTML语言中，你在页面中就可以使用它们，而不必为此修改文档类型声明。

由此，不少读者可能都会提出一个问题：既然HTML5是一门活语言，那为什么还要求网页中有这个所谓的文档类型声明呢？

要求保留文档类型声明，主要是由于历史原因。如果没有文档类型声明，那大多数浏览器（包括Internet Explorer和Firefox）将转换到一种混杂模式（quirk mode）[1]。在这种模式下，浏览器会尝试根据有点不那么正常的规则呈现网页（那些规则是在浏览器的老版本中使用的）。而问题是，不同浏览器的混杂模式也不一样，因此为一种浏览器设计的页面到了另一个浏览器中，不是字体大小不一样，就是布局上有瑕疵，或者出现其他不一致的问题。

而在添加了文档类型声明后，浏览器就知道你想要使用更严格的标准模式（standard mode），在这种模式下，所有现代浏览器都会以一致的格式和布局来显示网页。浏览器不关心你使用的是哪种文档类型（个别情况下还有些例外），只要它检查到你有某种文档类型声明就好。HTML5的文档类型声明是最短的有效文档类型声明，因此它总是能触发标准模式。

提示 HTML5的文档类型声明可以触发所有具备标准模式的浏览器的标准模式，包括那些对HTML5一无所知的浏览器。为此，从现在开始，你可以在任何网页中都使用HTML5文档类型声明，即便使用很少得到支持的HTML5功能也没问题。

虽然文档类型声明主要的目的是告诉浏览器去做什么，但其他代理也可以检测该声明，比如，HTML5验证器、搜索引擎、设计工具，还有人——在想知道你当初在页面中想写什么样的标记时。

注1：IE5.5引入了文档模式（document mode）的概念，这个概念是通过切换文档类型声明实现的。最初的两种文档模式是：混杂模式和标准模式。混杂模式会让IE的行为与（包含非标准特性的）IE5相同，而标准模式则让IE的行为更接近标准行为。虽然这两种模式主要影响CSS内容的呈现，但在某些情况下也会影响到JavaScript的解释执行。摘自《JavaScript高级程序设计（第3版）》P16 ～ P17，[美]Nicholas C. Zakas著，李松峰、曹力译，人民邮电出版社2012年版。（译者注）

1.3.2 字符编码

字符编码是一种标准，计算机根据它把文本转换成保存在文档中的字节序列（或者在打开文件时再将字节序列转换成文本形式）。由于历史原因，现有的编码标准有很多种。但实际上，所有英文网站今天都在使用一种叫UTF-8的编码，这种编码简洁、转换速度快，而且支持任何你想要的非英文字符。

一般来说，经过配置的Web服务器会告诉浏览器它提供的网页采用了什么编码。但除非是你自己配置的Web服务器，否则这一步始终是不确定的。由于浏览器在猜测网页编码的时候可能会引发一些说不清的安全问题，因此最好在自己的标记中也加上编码信息。

在HTML5文档中添加字符编码信息也很简单。只要像下面这样在<head>区块的最开始处（如果没有添加<head>元素，则是紧跟在文档类型声明之后）添加相应的<meta>元素即可：

```
<head>
  <meta charset="utf-8">
  <title>A Tiny HTML Document</title>
</head>
```

Dreamweaver等设计工具在创建新网页时会自动添加这个元信息，它们也会确保将文件保存为UTF编码格式。不过，如果你使用的是简单的文本编辑器，那就还要自己选择将文件保存为正确的格式。比如，使用Windows中的记事本程序编写HTML页面后，必须在“保存为”对话框下方的“编码”列表中选择“UTF-8”。而在使用Mac OS中的TextEdit时，首先需要选择“格式”➤“生成纯文本”，以确保程序将页面保存为纯文本，然后必须再从“保存为”对话框的“纯文本编码”弹出菜单中选择“Unicode(UTF-8)”。

1.3.3 页面语言

指明网页中使用的*自然语言*是一种好习惯。这个信息有时候对其他人有用，比如搜索引擎可以通过它来筛选搜索结果，确保只向搜索者返回页面语言与他使用的语言相同的页面。

为给内容指定语言，可以在任何元素上使用lang属性，并为该属性指定相应的语言代码（比如，en表示英语）。各国的语言代码可以在这里查到：http://people.w3.org/rishida/utils/subtags/。

为整个页面添加语言说明的最简单方式，就是为<html>元素指定lang属性：

```
<html lang="en">
```

如果页面中包含多种语言的文本，那么这个细节信息对屏幕阅读器也是很有用的。在这种情况下，可以为文本中的不同区块指定lang属性，指明该区块中文本的语言（例如，可以给包含不同语言文本的<div>元素指定不同的lang属性）。这样，屏幕阅读器就可以选择朗读适当的文本了。

1.3.4 添加样式表

只要是经过特意设计的专业网站，就一定会使用样式表。指定想要使用的CSS样式表时，需要在HTML5文档的<head>区块中添加<link>元素，例如：

```
<head>
  <meta charset="utf-8">
  <title>A Tiny HTML Document</title>
  <link href="styles.css" rel="stylesheet">
</head>
```

这跟向传统的HTML文档中添加样式表大同小异，但稍微简单一点。

> **注意** 因为CSS是网页中唯一可用的样式表语言，所以网页中过去要求的type="text/css"属性就没有什么必要了。

1.3.5 添加JavaScript

JavaScript最早是为了给网页添加一些闪光点和吸引力才出现的，不过编写起来比较费时间。今天，JavaScript的主要用途不再是装饰用户界面，而是开发新奇的Web应用，包括在浏览器中运行的极其先进的电子邮件客户端、文字处理程序，以及地图引擎。

在HTML5页面中添加JavaScript与在传统页面中添加差不多，下面就是一个引用外部JavaScript代码文件的示例：

```
<head>
  <meta charset="utf-8">
  <title>A Tiny HTML Document</title>
  <script src="scripts.js"></script>
</head>
```

没有必要加上language="JavaScript"属性。浏览器会假定你想要使用JavaScript，除非你想使用其他脚本语言，因为JavaScript是唯一被浏览器广泛支持的HTML脚本编写语言，所以你不会指定其他语言。不过，即使是引用外部JavaScript文件，也不能忘了后面的</script>标签。假如你不写这个标签，或者使用空元素语法想缩短标记，页面将不会执行加载脚本。

如果你在Internet Explorer中要花大量时间测试包含JavaScript的页面，还应该在<head>区块中包含一行特殊的注释，叫做Web标志（mark of the Web）[1]；这行注释要放在指定字符编码的元数据标签后面，如下所示：

```
<head>
  <meta charset="utf-8">
  <!-- saved from url=(0014)about:internet -->
  <title>A Tiny HTML Document</title>
  <script src="scripts.js"></script>
</head>
```

这条注释告诉Internet Explorer将页面视为从远程网站上下载下来的。否则，IE会切换到一种特殊的锁定模式，弹出一条安全警告，在你点了“允许阻止的内容”按钮之后才会执行JavaScript代码。

其他所有浏览器都会忽略这个“Web标志”注释，对远程站点和本地文件使用相同的安全设置。

注1：参见“Mark of the Web”：http://msdn.microsoft.com/zh-cn/library/ms537628(v=vs.85).aspx。（译者注）

1.3.6 最终结果

如果你按照上面这些步骤做了，那就有了一个如下所示的HTML5文档：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>A Tiny HTML Document</title>
  <link href="styles.css" rel="stylesheet">
  <script src="scripts.js"></script>
</head>

<body>
  <p>Let's rock the browser, HTML5 style.</p>
</body>
</html>
```

虽然这不再是一个最短的HTML5文档，但以它为基础可以构建出任何网页。这个例子本身没什么可圈可点之处，不过在下一章创建真实网页的过程中，我们会为内容精心设计布局并通过CSS应用样式。

注意 本节介绍的所有HTML5语法——新的文档类型声明、声明字符编码的元数据元素、语言信息属性和引用样式表及JavaScript标签，同时适用于新旧浏览器。因为这些语法依赖于所有浏览器的默认行为和内置的纠错机制。

1.4 HTML5 语法

如前所述，HTML5放松了某些规则。这是因为HTML5的制定者想让这门语言更紧密地反映浏览器的现实。换句话说，他们想缩小“可以工作的网页”与“根据标准是有效的网页”之间的差距。接下来，我们就介绍一下HTML5改变的语法规则。

注意 没错，还有很多浏览器支持的老式做法被HTML5标准严格排除在外。要想在自己的见面中及时改正这些老毛病，需要用到1.4.2节介绍的HTML5验证器。

1.4.1 放松的规则

在初次体验HTML5文档之后，我们知道HTML5并不要求网页中必须包含`<html>`、`<head>`和`<body>`元素（尽管它们的存在有时候非常有用）。但HTML5的轻松态度还不止于此。

HTML5不区分大小写，因此类似下面这样的标记是没有问题的：

```
<P>Capital and lowercase letters <EM>don't matter</eM> in tag names.</p>.
```

HTML5还允许省略关闭空元素（void element）的斜杠；所谓空元素，就是不会嵌套内容的元素，如<img>（图像）、
（换行）或<hr>（水平线）。以下是三种添加换行的等价方式：

```
I cannot<br />
move backward<br>
or forward.<br/>
I am caught
```

HTML5也修改属性的语法规则。属性值中只要不包含受限的字符（比如>、=或空格），就可以不加引号。下面这个<img>元素就利用了这一点：

```
<img alt="Horsehead Nebula" src=Horsehead01.jpg>
```

只有属性名没有属性值也可以。虽然XHTML要求必须采用如下冗余的语法将复选框设置为选中状态：

```
<input type="checkbox" checked="checked" />
```

但现在可以只包含属性名，回到HTML 4.01时代的传统短语法形式：

```
<input type="checkbox" checked>
```

对某些人来说，HTML5最令人担心的还不是这些。他们担心那些经常改变风格的开发人员会在严格的和松散的语法之间摇摆不定，特别是在一个文档内部也经常转换风格。可是，这种情况在XHTML时代同样存在。无论是严格还是松散，良好的风格都取决于Web设计师，而浏览器则会无条件地接受你扔给它的任何东西。

如果能做到如下几点（也是本书后续示例遵循的约定——尽管不是必须遵循的），基本上就可以算作良好的HTML5风格了。

- **包含可选的<html>、<body>和<head>元素**。要给页面定义自然语言（见1.3.3节），<html>是最理想的地方；而<body>和<head>有助于将页面内容与其他页面信息分离。
- **标签全部小写**（如用<p>而非<P>）。虽然不是必须这么做，但这种形式很常见，输入起来要轻松容易得多（因为不需要按Shift键），而且不会让人触目惊心。
- **为属性值加引号**。加引号是有理由的——防止你在不经意间犯错。要知道，没有引号的话，一个无效字符就可能破坏整个页面。

不过，还有一些老的约定这里并没有列出来（你也可以忽略）。本书的示例不会关闭空元素，因为在基于HTML5编写代码时，大多数人都不屑于添加额外的斜杠（/）。类似地，在属性名与属性值相同的情况下，还一味地留恋长属性的形式也没有什么道理。

1.4.2 HTML5 验证

没准儿新的松散的HTML5语法让你觉得很舒服。没准儿，一想到欢快的浏览器背后可能隐藏着不一致的、到处都是错误的标记，你简直夜不能寐。如果你不幸是后一种情况，那么一定会高兴听到这个消息：有验证工具可以帮你抓住那些与HTML5推荐标准不相符的标记，甚至都不会惊扰浏览器。

以下就是HTML5验证器会关注的一些可能的问题：

- ❑ 缺少必需的元素（例如<title>元素）；
- ❑ 有开始标签但没有结束标签；
- ❑ 标签嵌套错误；
- ❑ 不包含必要属性的标签（例如没有src属性的<img>元素）；
- ❑ 元素或内容放错了地方（例如把文本直接放在了<head>区块中）。

Dreamweaver等Web设计工具都有它们自带的验证器。如果你嫌麻烦，也可以使用在线验证工具，下面我们展示如何使用W3C标准组织提供的流行的验证器。

(1) 在浏览器中，打开http://validator.w3.org（图1-2）。

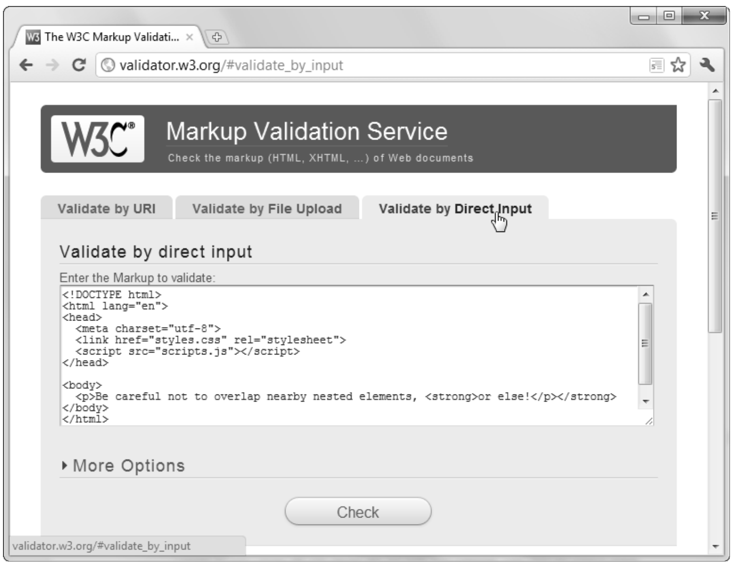


图1-2：W3C的验证器站点http://validator.w3.org提供了三个验证HTML的选项，可以填写网页的地址、上传网页或直接输入标记（如图所示就是直接输入标记）

(2) 单击相应的选项卡，提供HTML内容。

- ❑ “Validate by URI”可以验证已经存在的网页。只要在Address（地址）框中输入页面的URL即可（例如http://www.mysloppysite.com/FlawedPage.html）。
- ❑ “Validate by File Upload”可以验证你电脑硬盘上的页面。首先，单击Browse（浏览）按钮（在Chrome中单击Choose File<选择文件>）。在“打开”对话框中，选择HTML文件并单击Open（打开）。
- ❑ “Validate by Direct Input”可以验证任何标记——只要在大文本框里输入即可。从文本编辑器中直接把标记复制粘贴到W3C验证页面的文本框里是最简单的方式。

在继续之前，可以单击More Options（更多选项）修改设置，不过一般不用。最好是让验证器自动检测文档类型，因为这样验证器会使用你在网页中指定的文档类型。类似地，除非你的页面中使用了不同的语言，而验证器无法确定正确的字符集，否则就用自动检测好了。

(3) 单击Check（检测）按钮。

这样就会把HTML页面发送到W3C验证器。稍微等一会儿，就会看到报告。你会看到自己的文档是否通过了验证，而如果失败的话，则会看到验证器检测到的错误（见图1-3）。

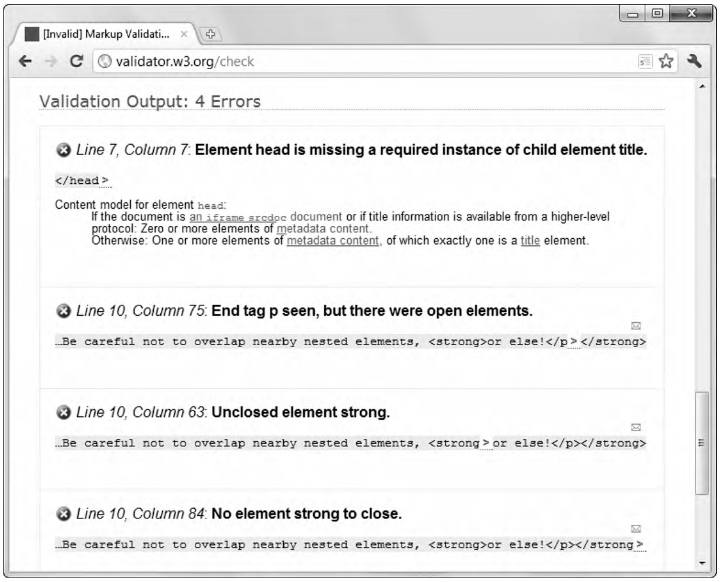


图1-3：验证器发现了由两个过失衍生出来的四个差错。首先，页面中没有必需的<title>元素；其次，<p>元素在嵌套的结束之前先结束。（要解决这些问题，把</p>替换成</p>即可。）顺便说一句，这个文档中的问题并不影响所有浏览器正确地显示它

注意 即便是验证之后没有发现一点问题的文档，验证器也会给出一些警告，包括字符编码是自动检测到的，HTML5验证服务还处于试验阶段、还不完善，等等。

1.4.3 XHTML的回归

如前所述，HTML5宣布了上一个Web王朝——XHTML时代的终结。但是，现实可没有那么简单，XHTML的拥趸也不必放弃上一代标记语言中自己最热爱的东西。

首先，别忘了XHTML语法还在呢。XHTML强制要求的规则要么仍具有指导意义（例如，元素要正确嵌套），要么仍然是一种得到支持的可选约定（例如，空元素可以包含结束的斜杠）。

要是想强制使用XHTML语法呢？也许你担心自己（或者自己的同事）在不经意间“堕落”到使用过去HTML的松散语法。不要紧，可以使用XHTML5；这个标准还没有多少人知道，其本质是给HTML5加上了XML的限制。

如果想把一个HTML5文档转换成XHTML5文档，必须在<html>元素中明确添加XHTML命名空间、关闭每一个元素、所有标签都要小写……下面就是一个XHTML5文档的示例：

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
  <meta charset="utf-8"/>
  <title>A Tiny HTML Document</title>
  <link href="styles.css" rel="stylesheet"/>
  <script src="scripts.js"></script>
</head>

<body>
  <p>Let's rock the browser, XHTML5 style.</p>
</body>
</html>
```

这样，就可以使用XHTML5验证器基于早先的XHTML规则对其进行更严格的错误检测了。标准的W3C的验证器不行，但可以使用http://validator.w3.org/nu，这个验证器在Preset（预设）下拉列表中提供了XHTML5选项。如果你不是直接输入页面的标记，或者不是把标记粘贴到文本框中，那还需要选上"Be lax about content-type"（不严格要求content-type）。

按照上述步骤，你可以创建并验证一个XHTML文档。可是，浏览器仍然只会将你的页面当成HTML5文档来解释，只不过这个文档有意要向XML靠拢罢了。除此之外，浏览器不会应用任何规则。

如果你想系统地支持XHTML5，还必须配置Web服务器，以application/xhtml+xml或application/xml的MIME类型来提供网页（不能再使用text/html；有关MIME类型请参见5.3.1节）。不过，在致电主机托管公司之前，务必清醒地认识到：这一修改会导致IE9之前所有版本的Internet Explorer均无法显示你的页面。换句话说，真正的XHTML5有浏览器兼容性问题。

有时候，即便支持XHTML5的浏览器在处理XHTML5文档时也会与处理普通的HTML5文档有所区别。这些浏览器将它作为XML文档处理，如果处理失败（比如因为你有个地方没写对），浏览器就不会再解释文档的其他部分了。

到底什么时候使用XHTML5？对于绝大多数Web开发人员，无论是一般人还是HTML5的铁杆粉丝，都没有必要使用XHTML5，以免招惹麻烦。唯一的例外，就是那些以XML作为开发目标的程序员。例如，想要使用XQuery和XPath等XML相关的标准来操作页面内容的开发人员。

提示 对于好奇的读者，我可以告诉你一个技巧，能让浏览器切换到XHTML模式——只要把文件的扩展名改为.xhtml或.xht即可。然后在硬盘中打开这个文件，多数浏览器（包括Firefox、Chrome和IE9+）都会认为该页面是从Web服务器下载下来的，而且MIME类型为XML。如果页面中有什么错误，浏览器窗口会显示只处理了一部分的页面（IE）、XML错误消息（Firefox）或二者的组合（Chrome）。

1.5 HTML5 元素家族

到目前为止，本章集中讨论了HTML5语法的变化，但更重要的则是新增、减少及改变了HTML哪些支持的元素。接下来的几节将分别讨论这些方面的变化。

1.5.1　新增的元素

在接下来的几章，我们将主要把时间放在学习新元素上，这些元素在此之前从来没有在网页中出现过。表1-1列表出这些新元素（以及哪一章包含对相应元素的详细介绍）。

表1-1　HTML5新增的元素

类　　别	元　　素	哪部分详细介绍
用于构建页面的语义元素	`<article>`、`<aside>`、`<figcaption>`、`<figure>`、`<footer>`、`<header>`、`<nav>`、`<section>`、`<details>`、`<summary>`	第2章
用于标识文本的语义元素	`<mark>`、`<time>`、`<wbr>`（以前就支持，但现在已经正式列入规范）	第3章
Web表单及交互	`<input>`（不是新元素，但增加了很多了类型）、`<datalist>`、`<keygen>`、`<meter>`、`<progress>`、`<command>`、`<menu>`、`<output>`	第4章
音频、视频及插件	`<audio>`、`<video>`、`<source>`、`<embed>`（以前就支持，但现在已经正式列入规范）	第5章
Canvas	`<canvas>`	第8章
非英语支持	`<bdo>`、`<rp>`、`<rt>`、`<ruby>`	HTML5 规 范 http://dev.w3.org/html5/markup

1.5.2　删除的元素

HTML5一方面添加了新元素，另一方面也从官方标准中剔除了少量元素。这些元素仍然可以得到浏览器支持，但任何遵循规范的HTML5验证器都会敏感地查出它们的藏身之所，并给出错误提示。

最明显的一点是，HTML5沿袭了不欢迎*表现性元素*的思想（最初萌发于XHTML）。所谓表现性元素，指的是那些仅仅是为网页添加样式的元素，而连最菜的Web设计人员也知道那是样式表该干的事儿。被剔除的元素都是专业开发人员很多年没有用过的元素（如`<big>`、`<center>`、`<font>`、`<tt>`和`<strike>`）。HTML的表现性属性也与之“同归于尽”了，没有什么必要在这里列出来了。

此外，HTML5进一步埋葬了Web开发人员原来已经摒弃的HTML框架。最初，HTML框架似乎是在浏览器窗口中显示多个网页的不错方式，但如今，框架往往意味着严重的可访问性问题，因为它们难以适应搜索引擎、辅助软件和移动设备。而有意思的是，`<iframe>`元素（通过它可以将一个网页放在另一个网页中）倒是侥幸得以保留。究其原因，主要是Web应用经常要利用`<iframe>`实现一些集成任务，比如在网页中包含YouTube窗口、广告和谷歌搜索框等。

还有另外一些元素，由于冗余或容易导致误会等原因也被剔除了，比如`<acronym>`（代之以`<abbr>`）和`<applet>`（因为`<object>`更好）。但元素家族的绝大部分成员照旧还生活在HTML5时代。

注意　HTML5元素家族中保留的元素超过100个。除此之外，差不多有30个新元素，还有大约10个有显著改变。要了解完整的元素列表（以及哪些是新元素、哪些改变了），请参考http://dev.w3.org/html5/markup/。

1.5.3 改变的元素

HTML5还有另一个奇怪的做法：有时候会将一个旧元素用于新的目的。例如，<small>元素的用途不再是减少文本字体的大小（这本来应该是样式表的任务）。HTML5虽然删除了<big>元素，但却保留了<small>元素，不过含义变了。现在，<small>元素表示“附属细则”（small print），比如页面底部没人想让你看到那些法律条款：

```
<small>The creators of this site will not be held liable for any injuries that may result from unsupervised
unicycle racing.</small>
```

放在<small>元素中的文本仍然照常显示，只不过字体稍小一点，除非你可使用样式表重写它的样式。

注意 人们对<small>元素有两种看法。一种看法认为它做到了最大限度的向后兼容，因为老浏览器都支持<small>元素，而HTML5页面中还将继续支持它。另一种看法认为它会导致旧网页中相应元素的语义变化，过去是用<small>元素来减少文本大小，但其中的文本不一定就是“附属细则”之类的。

另一个改变的元素是<hr>（horizontal rule，水平线），用于在两个区块间画一条线。在HTML5中，<hr>表示主题的转换，即从一个主题变为另一个主题。默认的格式还在，只不过又赋予了新的含义。

类似地，<s>（struck text，删除的文本）也不仅仅是给文本加一条删除线那么简单了，它现在表示不再准确或不再相关的内容。<hr>元素与<s>元素的变化都不及<small>元素那么大，至少还与它们传统HTML中的用法有联系。

粗体和斜体

要说最重要的变化，非粗体和斜体这两个格式化元素莫属。XHTML 1.0诞生后，HTML中最常用的两个表示粗体和斜体的元素<b>和<i>部分被<strong>和<em>元素取代。其背后的思想是停止从格式（粗体和斜体）的角度来看问题，而是要换成使用具有真实逻辑含义（重要或重音）的元素。这种思想当然很有意义，但<b>和<i>这两个标签仍然作为XHTML新引入的两个标签的简写形式存在着，因为大家对它们更熟悉。

HTML5尝试解决这个问题。它没有强迫开发人员放弃<b>和<i>，而是为这两个元素赋予了新的含义。背后的思想就是允许上述4个元素在有效的HTML5文档中共存，但结果多少让人有点迷惑，下面分别说明。

- 使用<strong>表示重要的文本内容，也就是那些需要在周围文本中突出出来的文本。
- 使用<b>表示应该用粗体表示的文本，但该文本并不比其他文本更重要。比如，关键字、产品名称等所有需要用粗体表示的文本都可以用这个标签。
- 使用<em>表示重读的文本，也就是在朗读的时候要大声读出来。
- 使用<i>表示应该用斜体表示的文本，但该文本并不比其他文本更重要。比如，外文单词、

技术术语等所有需要使用斜体表示的文本都可以用这个标签。

以下这段代码以适当的方式使用了上述全部4个标签：

```
<strong>Breaking news!</strong> There's a sale on <i>leche quemada</i> candy
at the <b>El Azul</b> restaurant. Don't delay, because when the last candy
is gone, it's <em>gone</em>.
```

在浏览器中，会看到如下结果：

Breaking news! There's a sale on *leche quemada* candy at the **El Azul** restaurant. Don't delay, because when the last candy is gone, it's *gone*.

对Web开发人员来说，他们有的会遵循HTML5这个善意的规则，有的还是继续使用自己最熟悉的元素去标粗体和斜体格式。

1.5.4 调整的元素

HTML5也调整了一些元素的使用规则。要说啊，这些调整只有那些死钻HTML的家伙才会注意到，但偶尔也会产生比较大的影响。举一个例子，不太常用的<address>元素并不适合标注邮政地址（尽管address是“地址”的意思）。实际上，这个元素只有一个目的，即提供HTML文档作者的联系信息，比如电子邮件地址或网站链接：

```
Our website is managed by:
<address>
<a href="mailto:jsolo@mysite.com">John Solo</a>,
<a href="mailto:lcheng@mysite.com">Lisa Cheng<a>, and
<a href="mailto:rpavane@mysite.com">Ryan Pavane</a>.
</address>
```

再比如，<cite>元素的含义也不一样了。当然，像下面这样引用某些作品（如新闻、文章、电视节目）还是可以的：

```
<p>Charles Dickens wrote <cite>A Tale of Two Cities</cite>.</p>
```

可是，现在用<cite>去标注人名已经不对了。这个变化最终导致了令人意想不到的争议，因为以前是可以这么用的。一些“骨灰”级的Web开发人员公开发表言论，鼓动人们不用遵守<cite>的新使用规则。这可真让人有点匪夷所思，毕竟你一辈子能在编辑网页的时候看见几回<cite>元素啊？

对用于创建链接的<a>元素的调整幅度相对更大一些。HTML以前的版本允许用<a>元素来标注可以单击的文本或图像。而在HTML5中，可以在<a>元素中放置任何东西；就是说，你在里面放上一大段文字、一个列表、几幅图像……都没有问题。（如果你真这么做，那就会发现里面的所有文本都会变成蓝色并带有下划线，而图像则会产生蓝色的边框。）Web浏览器支持这种做法已经有很多年了，HTML5只不过是把这种行为写进了规范，即便这种用法没什么大用处，但毕竟已经列入了标准。

还有一些调整在目前所有浏览器中都还未得到支持。例如，<ol>元素（有序列表）现在有了一个reversed属性，用于反转序号（直到1或其他通过start属性设置的正序时的起始值）。不过，

这个属性目前只有Chrome和Safari支持。

在学习本书的过程中，我们还会陆续介绍其他一些被调整了使用规则的元素。

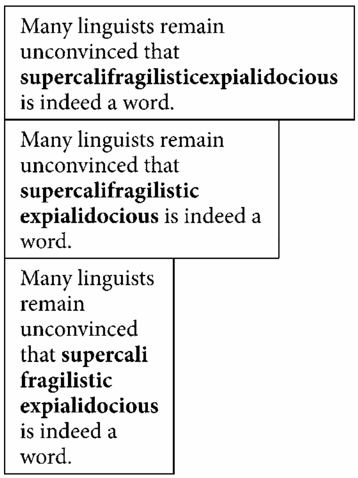
1.5.5 标准化的元素

HTML5还把一些浏览器实际支持，但并没有得到之前的HTML或XHTML规范承认的元素加入了标准。其中最广为人知的一个元素就是<embed>，这个元素在目前的网页中得到了普遍使用，成为了一种向页面中加入插件的通用方法。

另一个新元素是<wbr>，表示可以在某处断行。换句话说，如果某个词太长了，一行放不下，那浏览器就会在<wbr>标注的地方断开：

```
<p>Many linguists remain unconvinced that
<b>supercali<wbr>fragilistic<wbr>expialidocious</b> is indeed a word.</p>
```

如果需要在小空间（如表格单元或小方块）里放长名字（比如编程语言中的变量名），可以用<wbr>来标注可以在名字的什么地方断行。不过，即使浏览器支持<wbr>，它也只会在相应空间盛不下长名字时才会断行。对于前面的标记，在浏览器中有可能看到以下三种情况：



说到<wbr>，难免会让人想起类似的<nobr>元素，后者用于阻止文本换行，可用空间再小也不行。但HTML5认为<nobr>已经不合时宜，建议所有自重的Web开发人员不要再使用它了。要想达到同样的效果，可以在CSS中使用`white-space`属性，将它的值设定为`nowrap`。

1.6 今天开始用 HTML5

在使用HTML5之前，必须知道你的用户浏览器对它的支持情况。毕竟，我们希望得到的是一个耀眼的新页面，而非一堆混乱的标记和遇上过时浏览器就错误百出的脚本。

稍后，我们会介绍怎么知道HTML5的哪个具体功能得到了哪些浏览器的支持，以及你的用户

中有多大比例在使用这些浏览器。不过，在此之前，我们先概括一下HTML5当前受支持的情况。

- 如果你的用户使用流行的Chrome或Firefox，总体来说问题不大。这两款浏览器不仅已经支持HTML5很多年了，而且都能够自动升级。换句话说，不大可能有这两款浏览器的老版本存在。
- 如果你的用户使用Safari或Opera，那也还好。这两款浏览器同样支持HTML5很多年了，同样也没有多少老版本用户。
- 如果你的用户使用平板电脑或智能手机，那可能某些功能会有一些限制，本书后面还会介绍到。不过，所有今天的移动浏览器在设计时都会考虑支持HTML5，因此你的网页偶尔可能出现点问题，但总体还可以接受。
- 如果你的用户使用旧版本的IE，也就是IE10之前的任何版本，那大多数HTML5功能都将无法使用。问题在哪儿呢？因为如今很多旧版本Windows仍然常见，这些操作系统都预装了旧版本IE。更糟糕的是，很多旧版本Windows会阻止用户升级到现代的支持HTML5的IE。比如Windows Vista只允许用户使用IE9，而在令人谈之色变的Windows XP上则只能用IE8。

并非微软想要阻挠Web的发展，而是新版本IE都是针对新硬件设计的。换句话说，新软件放到旧硬件上就是不能运行。然而，使用旧版Windows的用户还可以选择Firefox替代IE，只不过有人不知道怎么安装Firefox，或者不被允许这么干罢了。

注意 尽管IE6和IE7已经基本退出了历史舞台，但（写作本书时）IE8和IE9的流量仍然有百分之十以上。我们当然不能忽视百分之十的用户，因此必须想办法为大多数HTML5功能寻找替换手段，至少在不远的将来还是需要这样做。

对付旧版本的浏览器

接下来的几年，一些用户的浏览器可能仍然不会支持你要使用的全部HTML5功能。这是我们要面对的现实。不过，只要你不介意麻烦点儿，那也不是问题。这时候有两种处理方式。

- **平稳退化**。有时候，如果某个功能不被浏览器支持，也不是什么大问题。例如，`<video>`元素提供了一种后备机制，可以为老的浏览器提供替代内容（比如使用Flash插件的视频播放器）。只显示一条错误消息就很不礼貌了，不能成为平稳退化的模范！还有一些表单功能（例如占位符文本），以及像圆角和投影之类的CSS3属性，较老的浏览器完全可以忽略它们。当然，可以写一些JavaScript代码来检测浏览器是否支持你想使用的功能（可以使用Modernizr）。如果不支持，则显示较少内容或使用不那么花哨的方式。
- **借助JavaScript**。HTML5的很多新功能都源自Web开发人员费尽心机做出来的东西。因此，HTML5中的某些功能完全可以使用优秀的JavaScript库来实现（最差的情况，完全通过手工编写JavaScript代码也可以写出来），这一点也不奇怪。通过JavaScript来实现这些功能很费事儿，不过网上已经有很多写得很好（或不那么好）的解决方案了，你可以直接拿来使用。其中一些比较成熟的则称为“腻子脚本”（参见1.6.4节）。

1.6.1 了解浏览器支持情况

到底能使用哪些HTML5的功能，最终还是由浏览器开发商说了算。如果它们不支持某个功能，无论标准里怎么说，最终还是不能用。今天，主流的浏览器差不多有四五个（不包括智能手机、平板电脑等能上网的设备中的移动浏览器）。作为个体，没有几个Web开发人员能自己完全测试每个浏览器的每一项功能，更不用说检测那些还被很多人使用的老浏览器的支持情况了。

好在，有一个Can I use网站可以帮我们。这个网站可以详细地列出每一款主流浏览器对HTML5的支持情况。更重要的是，它还能让你针对实际需要的功能查询浏览器。下面是这个网站的使用指南。

(1) 在浏览器中打开http://caniuse.com。

可以看到，主页面中有很多分了组的链接，如CSS、HTML5，等等。

(2) 选择你想了解的功能。

找到功能的最简单方法，就是在顶部的搜索框里输入并搜索。

或者，也可以通过点击首页中的链接来查询某一功能。比如，HTML5链接组是HTML5标准中规定的功能；JS API组是那些依赖JavaScript的功能，这些功能一开始属于HTML5，但后来又被剥离了出来；而CSS组是属于CSS3的功能。

提示　点击每个组的链接，比如HTML5或JS API，可以一次性查看所有功能的受支持情况。

---

(3) 查看结果（图1-4）。

在每个功能的表格中，都会包含一组不同的浏览器版本。表格中用单元格的颜色表示对功能的支持情况：红色表示不支持，浅绿色表示支持，橄榄绿色表示部分支持，灰色表示不确定（一般是因为浏览器的当前版本还在开发过程中，相应功能还没有加入）。

(4) 查看更多浏览器细节。

起初，这个支持表格只包含流行浏览器的最新版本。而通过调整，也可以让它包含其他浏览器的支持情况，比如IE7或Android版移动Firefox浏览器。

要选择表格中包含哪个浏览器，首先单击表格上方的Show options链接。然后会出现一组浏览器复选框，选中复选框，可以添加浏览器。拖动Version shown中的滑块则可以设定阈值，根据需要排除不常用的浏览器版本。

另外，单击表格左上角的Show all version链接，则可以显示Can I use数据库里收集到的所有浏览器兼容信息。不过，这时候的表格会非常大，而显示的浏览器甚至包含古老的Firefox 2和IE5.5。

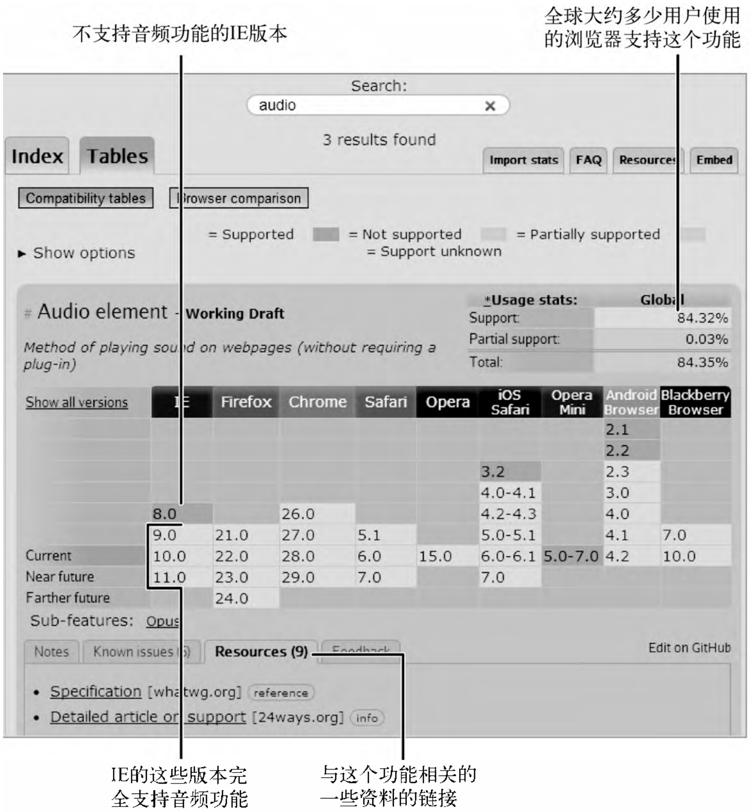


图1-4：搜索audio得到两个表格。第一个表格展示的是浏览器对<audio>元素的支持情况（图中所示）

绿色单元格中显示的是完全支持音频功能的浏览器版本号，红色单元格显示的是不支持音频功能的浏览器版本号

1.6.2　浏览器装机情况统计

怎么知道应该关注哪个浏览器版本呢？浏览器采用情况统计可以告诉你，有多少用户的浏览器支持你想使用的功能。为此，可以查看流行的流量跟踪站点GlobalStats，用法如下。

(1) 在浏览器中打开http://gs.statcounter.com。

打开网站后，可以看到一个折线图，显示着最近几年常用的浏览器。但此时的图表不包含版本信息，从中也看不出来有多少人还在使用IE的那些问题版本（IE10之前的版本）。要看到这些信息，还要调整设置。

(2) 找到Stat链接（图表左下方），点击打开设置项，选择Brower Version（Combine Chrome& Firefox）。

这样，不仅可以看到目前正在使用的浏览器，也能看到使用的是它们的哪个版本。组合Chorme和Firefox，是因为这两款浏览器更新太快，这样可以减少十几条折线，让图形简单直观（图1-5）。

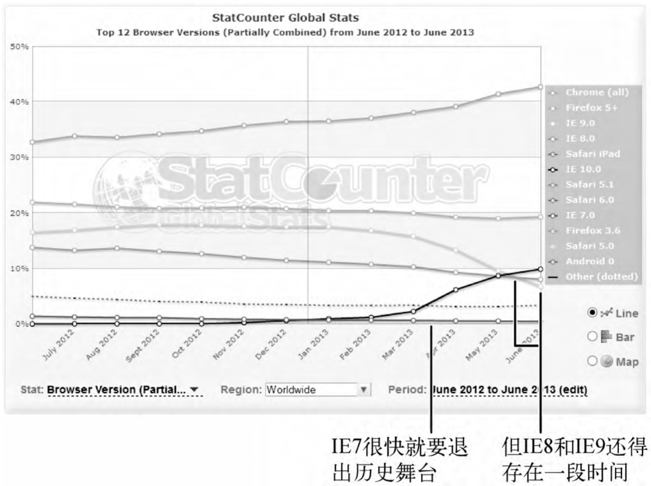


图1-5：如图所示，尽管Chrome如日中天，但IE8和IE9等问题浏览器仍然有一定的市场

(3) 在Region中选择地理区域。

默认设置是全球（Worldwide），即全世界的使用统计信息。可以在这里选择具体的国家（如Bolivia）或地区（如North America）。

(4) 在Period中选择日期范围。

一般来说，我们会关注一整年的浏览器使用率变化情况。但如有必要，也可以选择更短的时间范围，比如过去的三个月。

(5) 使用图表右侧的选项按钮切换图表类型。

选择Line，就是以折线图显示浏览器使用率随时间推移的变化情况。选择Bar，就是以柱形图显示当前浏览器使用率。选择Map，则以彩色地图显示某地区使用率最高的浏览器。

GlobalStats每天都通过它遍布数百万网站的跟踪代码来生成统计信息。虽然涉及的页面量巨大，数据量也非常大，但总归还是整个Web的一小部分。换句话说，你不能假定自己的网站访客都使用相同的浏览器。

更进一步，浏览器装机份额还会随用户所在国家和网站类型而变化。比如，德国40%以上的用户使用Firefox，位居第一。在访问TechCrunch网站（计算机极客最喜欢的新闻网站）的用户中，使用旧版本Internet Explorer的非常少。所以说，要想设计符合自己用户需求的网站，就必须以你自己的页面生成的统计信息作为依据。（如果你还没有在自己的站点中使用网页跟踪服务，我推荐你使用最好用而且免费的Google Analytics：www.google.com/analytics。）

1.6.3　通过Modernizr检测功能

检测功能是应对浏览器支持差异的一个重要策略。典型的模式是：加载页面，通过JavaScript脚本检测某个具体的功能是否可用。然后，可以对用户给出提示（最衰的做法）、平稳退化到没那么花哨的版本（稍好一些），或者采用其他方法实现浏览器本该支持的HTML5功能（最佳）。

遗憾的是，由于HTML5本质上是一个松散的相关标准的集合，因此不可能通过一次测试就能验证所有功能。相反，为了检测不同的功能，必须分别运行各种不同的测试——甚至，有时候还会测试浏览器是否支持某项功能的某个部分，而测试速度会非常快。

检测支持通常需要检查某个可编程对象的属性，或者创建一个对象并以特定的方式使用它。不过，在按照这种思路编写测试代码之前，一定要三思而行；因为弄不好，可能会非常麻烦。比如，由于种种原因，你的测试代码在某些浏览器上总是失败，或者过不了多久，又要重写测试代码。所以，我建议你使用Modernizr（www.modernizr.com），它是一个小巧的、持续更新的工具，专门用于测试浏览器对很多HTML5及相关功能的支持情况。如果你想使用新的CSS3功能，本书6.1.3节还将介绍一个实现后备支持的绝妙技巧。

在页面上使用Modernizr的方法如下。

(1) 在浏览器中打开http://modernizr.com/download。

找到Development version链接，这个链接指向最新的完整版Moderniznr。

(2) 右键单击Development version链接，选择“链接另存为”或“目标另存为”。

命令名不一样，但功能一样，具体名称取决于浏览器。

(3) 选择保存文件的目录，单击“保存”。

保存的JavaScript文件名为moderniznr-latest.js。

(4) 把下载到的文件放在你的网页所在的文件夹中。

或者，放在一个子文件夹中也可以，然后修改引用该JavaScript文件的路径。

(5) 在页面的<head>区块中添加对这个JavaScript文件的引用。

假定moderniznr-latest.js文件与你的网页放在同一文件夹中，下面就是代码片段的示例：

```
<head>
  <meta charset="utf-8">
  <title>HTML5 Feature Detection</title>
  <script src="modernizr-latest.js"></script>
  ...
</head>
```

这样，当页面加载后，Modernizr脚本就可以运行了。它能够在短短的数毫秒时间内检测很多新功能，然后创建一个名叫Modernizr的新JavaScript对象，检测结果就保存在这个对象里。通过检测这个对象的属性，就可以知道浏览器具体支持什么功能。

提示 要了解Modernizr能够检测的所有功能，以及需要检测的JavaScript对象的所有属性，请参考相关文档，地址为www.modernizr.com/docs。

(6) 编写脚本检测你想使用的功能，然后执行相应的操作。

例如，要检测浏览器是否支持HTML5的拖放功能，并将检测结果显示在页面上，可以使用以下代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>HTML5 Feature Detection</title>
  <script src="modernizr-latest.js"></script>
</head>

<body>
  <p>The verdict is... <span id="result"></span></p>

  <script>
// 找到页面中用以显示结果的元素（id为result）
var result = document.getElementById("result");
if (Modernizr.draganddrop) {
    result.innerHTML = "Rejoice! Your browser supports drag-and-drop.";
}
else {
    result.innerHTML = "Your feeble browser doesn't support drag-and-drop.";
}
  </script>
</body>

</html>
```

图1-6显示了结果。

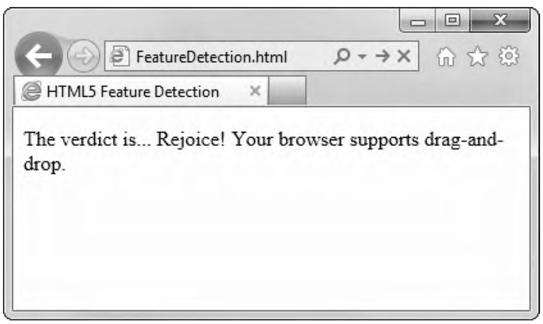


图1-6：虽然这个例子展示了检测功能的正确方法，但使用功能的方式不怎么理想。与其直接告诉你的访客他们的浏览器不支持这个功能，远远不如实现一个后备方案（即使结果不如使用HTML5功能那么好），甚至不如简单地忽略这个问题（如果相应的功能只是为了实现某个装饰性的效果，有没有它还不是无所谓）

提示　这个例子中使用了一种久负盛名的基本JavaScript技术——根据ID查找元素并修改其内容。如果你觉得这个例子不好理解，建议你先看一看附录B中关于JavaScript的入门知识。

完整的Modernizr脚本体量有点大，对于测试页面是没有问题的。但在完成开发后，应该创建一个只包含待检测功能的简化版Modernizr。为此，再次打开http://modernizr.com/download页面，但不是点击Development version链接，而是要选择复选框。把你想检测的功能依次选中后，点击GENERATE!按钮，就可以生成定制的Modernizr。再点击DOWNLOAD按钮把文件保存到本地硬

盘即可（图1-7）。

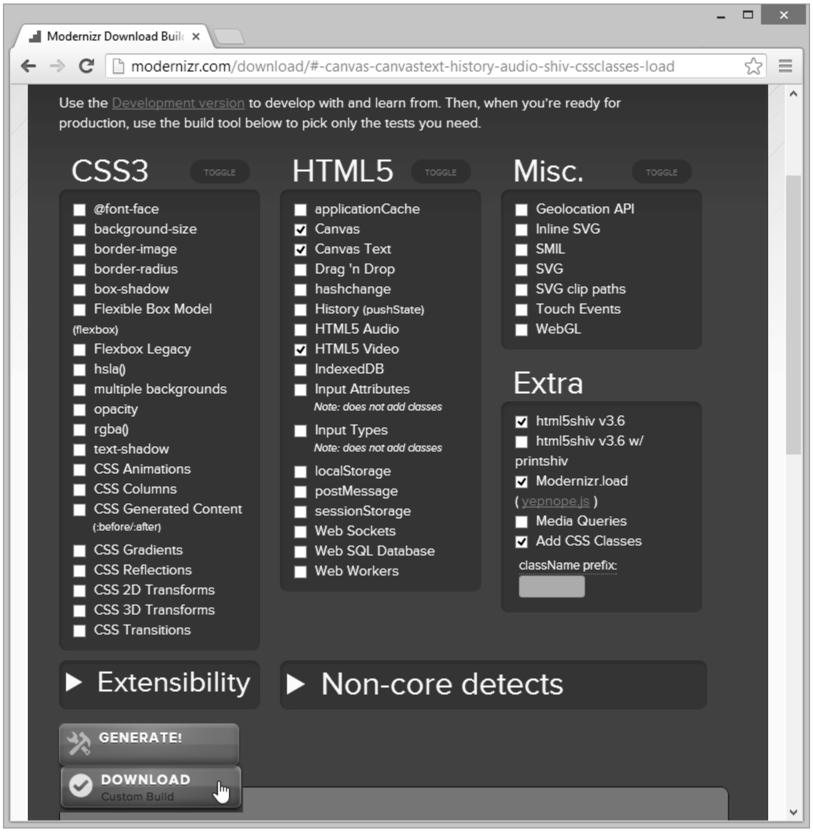


图1-7：假设你需要一个检测HTML5画布、画布文本及HTML5视频功能的版本，那么检测其他功能的代码就不会出现在生成的Modernizr文件中

1.6.4　使用“腻子脚本”填补功能缺陷

Modernizr可以帮你找出浏览器支持上的缺陷。它会在某个功能不可用时提醒你，但除此之外不会帮你弥补这些缺陷。而这正是我们要介绍的腻子脚本（polyfill）的用途所在。从根本上说，腻子脚本就是一大堆五花八门的技术，目的就是填平旧浏览器对HTML5支持上的缺陷。英文单词polyfill源自英国的一种腻子粉，而腻子粉就是在刷墙漆之前用来填补墙面裂缝和漏洞的（腻子粉在美国叫spackling paste）。在HTML5中，理想的腻子脚本可以直接放到页面中使用，不必多做额外的工作。然后，这些脚本就能无缝地保持向后兼容，而且一点都不唐突，让你在其他人苦思冥想变通方案时就能用上纯粹的HTML5。

不过，腻子脚本并不完美。有些腻子脚本依赖的技术同样得不到普遍支持。例如，有一个“腻子脚本”可以通过Silverlight插件在老版本的Internet Explorer中模拟HTML5的画布。而假如访客不愿意安装Silverlight，那你还得考虑一种后备方案。还有一些腻子脚本实现的功能比HTML5规

定的功能少一些，或者性能上要差一点。

有时候，本书会告诉你某个腻子脚本可以考虑。假如你想了解更多的信息，可以在GitHub上找到最相关的信息，乃至各种HTML5腻子脚本的完整集合，页面地址为：http://tinyurl.com/polyfill。不过我还要提醒一句：这些腻子脚本在品质、性能和支持等方面有着非常大的差异。

提示　光知道针对HTML5的某项功能有一个腻子脚本还不够。在自己的网站中实际地使用它们**之前**，必须先在各种老浏览器上测试，事先掌握它们的实际运行效果。

浏览器装机统计信息、功能检测再加上腻子脚本，在这些方面做足了功课之后，就可以考虑怎么应用HTML5功能了。下一章，我们将迈出第一步，介绍一些在新、旧浏览器中都能使用的HTML5元素。

第 2 章

用语义元素构造网页

从Web诞生到现在的20年间，网站发生了天翻地覆的变化。不过，最让人叹为观止的，还不是Web的变化有多大，而是最古老的HTML元素到今天依然被沿用着！事实上，Web开发人员在构建现代网站时使用的HTML元素，与10年前构建网站时使用的HTML元素别无二致。

有一个元素特别值得一提，那就是温和谦恭的<div>（division，分区）元素，它堪称每一个现代网页的柱石。利用<div>元素，可以把整个HTML文档分隔为页眉、侧边面板、导航条，等等。再辅以少量可靠的CSS，就可以把这些区块转换成带边框的盒子或带阴影的分栏，而且各就各位。

这种<div>加样式的技术既简明又强大，还非常灵活——但不够透明。在查看别人的源代码时，必须费点劲才能知道哪个<div>表示什么，而整个页面又是怎么搭建起来的。为了理解页面的构造，不得不在标记、样式表和浏览器显示的页面之间跳来转去。特别是在破解别人编写的不怎么符合最佳实践的页面时，即便你也在自己的网站中应用了同样的设计技术，也少不了会面对这种困惑。

这种情况引发了人们的思考。是不是可以用更好的东西来代替<div>？这种东西需要发挥与<div>一样的作用，但却能传达出更多的语义。而且，要能够把侧边栏与页眉分开，以及把广告条与菜单分开。HTML5为此引用了一组构造页面的新元素，实现了Web开发人员的这一夙愿。

提示 如果你的CSS技能尘封已久，急需温故知新，然后才能看懂样式表，那说明你还没法学习这一章。好在附录A中包含一个简明的CSS教程，在那里可以找到CSS知识的基本介绍。

2.1 语义元素

要想让网页的结构更清晰，需要使用HTML5中新的*语义元素*（semantic element）。这些元素可以为它们标注的内容赋予额外的含义。例如，新的<time>元素用于在网页中标注一个有效的日期或时间。下面就是<time>元素最简单的用例：

```
Registration begins on <time>2014-11-25</time>.
```

这行代码在网页中呈现的结果如下所示：

```
Registration begins on 2014-11-25.
```

最关键的是要理解，`<time>`元素没有任何内置的样式。实际上，网页的读者也没有办法知道有一个额外的元素包含了日期。你可以使用样式表为`<time>`元素添加样式，但默认情况下，`<time>`元素中的文本与普通文本没有任何区别。

2

设计`<time>`元素的用意是让它来包含一小段信息。不过，大多数HTML5语义元素的用途是标识页面中的一个内容区块。比如，`<nav>`元素用于标识一组导航链接。而`<footer>`元素用于标识通常放在页面底部的文脚（或页脚）。算起来，大概有十几个类似的新元素。

注意 尽管语义元素在HTML5的新功能里不怎么起眼，但它们的数量却不少。HTML5新增的大部分元素都是语义元素。

所有语义元素都有一个显著的特点：不真正做任何事。相对来说，`<video>`元素则囊括了在页面中充当视频播放器的全部能力（参见5.2.5节）。有读者可能会问了，既然它们不会改变网页的外观，那为什么还要使用这些新元素呢？

有如下几条理由。

- **容易修改和维护**。解读传统的网页比较困难。要想理解整体布局和不同区块的重要程度，必须得一遍一遍地看网页的样式表。但通过使用HTML5的语义元素，通过标记就可以传达出额外的结构化信息。这样，等你几个月后再回头修改网页，就不会像以前那么头疼了。当然，如果是其他人需要帮你改进页面，使用语义元素就显得更重要了。
- **无障碍性**。现代Web设计的一个重要主题，就是让任何人都能无障碍地访问网页。换句话说，要让使用屏幕阅读器和其他辅助工具的人都能在页面中自由导航。兼容HTML5的无障碍工具可以为残疾人士提供更好的上网体验。（仅举一个例子，有了`<nav>`元素，屏幕阅读器就能够迅速返回导航区，进而找到网站的链接。）

提示 要了解针对Web无障碍性的最佳实践，可以访问WAI（Web Accessibility Initiative，Web无障碍倡议）的网站：www.w3.org/WAI。或者，要了解通过屏幕阅读器上网是一种什么样的感觉（同时理解为什么标题要排列适当），可以看看YouTube的这段视频：http://tinyurl.com/6bu4pe。

- **搜索引擎优化**。像谷歌这样的搜索引擎，会使用强大的搜索机器人（search bot），这些搜索机器人自动在Web中爬行并获取每一个网页。然后扫描网页内容并将它们索引到搜索数据库中。如果谷歌能够更好地理解你的站点，那搜索者的查询就会越容易与你的内容匹配，因而你的网站列在搜索结果中的可能性也就越大。搜索机器人已经在检查一些HTML5的语义元素了，这样可以收集到它们索引的页面的更多信息。

- **未来的功能**。新浏览器和Web编辑工具一定会利用语义元素。比如，浏览器可以提供一个页面的内容纲要，以方便访客跳转到页面中适当的区块。（事实上，Chrome已经提供了一个显示页面纲要的插件，详见2.5.1节。）类似地，网页设计工具也能包含一些方便你构建或编辑导航菜单的功能，而方法就是组织你放在<nav>区块中的内容。

最关键的问题在于，如果你正确地使用了语义元素，就能够创建更加清晰的页面结构，就能够适应未来的浏览器和Web设计工具的发展趋势。而如果你还是抱着原来写HTML标记的老习惯不放，那就要跟未来擦肩而过了。

2.2 改造传统的 HTML 页面

要了解和熟悉新的语义元素（包括学习如何使用它们来构造页面），最好的方式莫过于拿一个经典的HTML文档作例子，然后把HTML5的一些新鲜营养充实进去。图2-1是我们要改造的第一个页面。这个页面很简单，只包含一篇文章，当然对于其他内容（如博客、产品说明或简短的新闻报道）也完全没有问题。

提示 访问http://prosetech.com/html5，可以查看或下载图2-1中的示例页面以及本章所有的示例页面。如果你想从头开始做，那就选择ApocalypsePage_Original.html，如果想直接查看使用HTML5改造之后的结果，请选择ApocalypsePage_Revised.html。

2.2.1 构造页面的老办法

要生成图2-1所示的页面，有很多种构造方法。让人高兴的是，这个示例页面使用的是HTML最佳实践，因此没有任何通过标记来进行格式化的痕迹。没有粗体或斜体元素，没有嵌入的样式，当然更没有土得掉渣的<font>之类的东西。总之，这是一个格式非常规范的页面，所有样式均来自外部样式表。

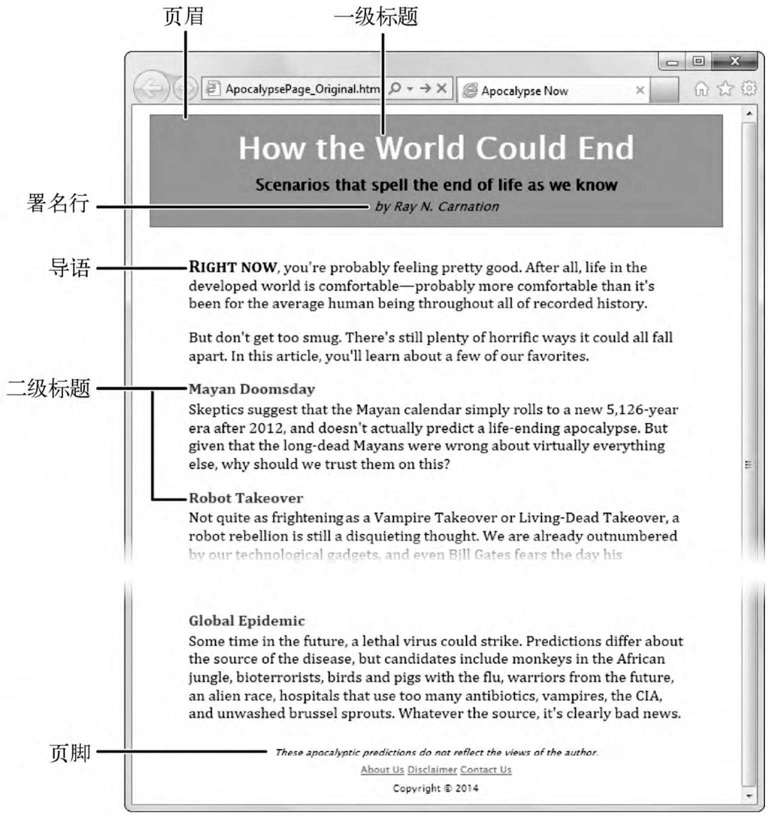


图2-1：一个普通的HTML页面，具有类似文档的页面结构，其中的格式来自一个外部样式表

以下是从页面中摘出的一段标记，代码中加粗的地方表明应用了样式：

```
 <div class="Header">
   <h1>How the World Could End</h1>
   <p class="Teaser">Scenarios that spell the end of life as we know it</p>
   <p class="Byline">by Ray N. Carnation</p>
</div>

 <div class="Content">
   <p><span class="LeadIn">Right now</span>, you're probably ...</p>
   <p>...</p>

   <h2>Mayan Doomsday</h2>
   <p>Skeptics suggest ...</p>
 ...
</div>

 <div class="Footer">
   <p class="Disclaimer">These apocalyptic predictions ...</p>
   <p>
     <a href="AboutUs.html">About Us</a>
```

```
      ...
    </p>
    <p>Copyright © 2014</p>
</div>
```

代码中的省略号是怎么回事

本书不可能把每个示例的所有标记都印出来，除非把它扩充到12 000页，再毁掉一片成材林。不过，这些代码**能够**展示出页面的基本结构，以及所有重要的元素。为此，代码中使用了省略号（...），用以表示省略未印刷出来的内容。

例如，就拿这段代码来说，其中包含了图2-2所示页面主体中的所有内容，只不过省略了一些包含文字的段落、Manyan Doomsday后面的条目以及页脚中的一些链接。如果你想仔细观察页面中的每一处细节，当然没有问题，就在本书试验站点（http://prosetech.com/html5）查看示例文件即可。

在一个（像这个一样的）编写规范的传统HTML页面中，通过使用`<div>`和`<span>`元素，已经把大部分工作移交给了样式表。通过`<span>`可以为处在其他元素中的少量文本添加样式，而通过`<div>`不仅可以为整个内容区块添加样式，还可以构建起整个页面的结构（见图2-2）。

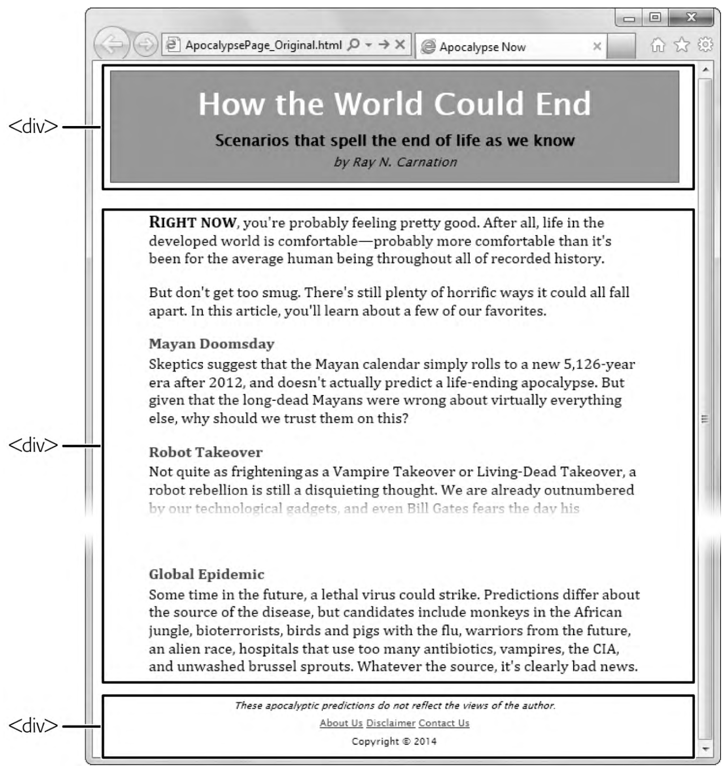


图2-2：通过`<div>`元素把页面分隔为三个逻辑区块：顶部的页眉、中部的内容和底部的页脚

这个例子中的样式表比较简单。整个页面的最大宽度设置为800像素，避免文本在宽屏显示器上显示得过长。页眉位于一个带蓝色边框的盒子中，内容区的两侧都添加了内边距，而页脚在整个页面底部居中。

因为使用了<div>元素，所以添加样式很容易。比如，样式表ApocalypsePage_Original.css使用下列规则为页眉及其中内容添加样式。

2

```
/*为<div>添加样式，使其具有页眉的外观（蓝色带边框）*/
.Header {
  background-color: #7695FE;
  border: thin #336699 solid;
  padding: 10px;
  margin: 10px;
  text-align: center;
}

/*为页眉中的<h1>添加样式（这是文章的标题）*/
.Header h1 {
  margin: 0px;
  color: white;
  font-size: xx-large;
}

/*为页眉中的子标题添加样式*/
.Header .Teaser {
  margin: 0px;
  font-weight: bold;
}

/*为页眉中的署名行添加样式*/
.Header .Byline {
  font-style: italic;
  font-size: small;
  margin: 0px;
}
```

你可能注意到了，这些规则有效利用了上下文选择符（见A.3.3节）。比如，用选择符.Header h1选择了页眉区域中的所有<h1>元素。

提示　附录A在介绍CSS时也用到了这个例子。如果你想知道应用给这个页面的所有样式规则，可以翻到A.4节。

2.2.2　使用HTML5 构造页面

<div>目前依旧是Web设计的必备元素，它是一个直观、多用途的容器，可以通过它为页面中的任何区块应用样式。但<div>的问题在于，它本身不反映与页面相关的任何信息。在你（或浏

览器、设计工具、屏幕阅读器、搜索机器人）遇到一个<div>元素时，你知道它是页面中独立的一个区块，可是你不知道那个区块的意图。

要通过HTML5改进这种情况，可以把<div>替换成更具有描述性的语义元素。这些语义元素的行为与<div>元素类似：它们仅包含一组标记，除此之外没有其他作用，可以将它作为"格式挂钩"来为页面应用样式。不过，除此之外，它们还会为页面添加一点语义。

下面的代码是在图2-1所示页面基础进行了简单修改之后的结果，删除了两个<div>元素，但添加了两个HTML5的语义元素：<header>和<footer>。

```
<header class="Header">
  <h1>How the World Could End</h1>
  <p class="Teaser">Scenarios that spell the end of life as we know it</p>
  <p class="Byline">by Ray N. Carnation</p>
</header>

<div class="Content">
  <p><span class="LeadIn">Right now</span>, you're probably ...</p>
  <p>...</p>

  <h2>Mayan Doomsday</h2>
  <p>Skeptics suggest ...</p>
  ...
</div>

<footer class="Footer">
  <p class="Disclaimer">These apocalyptic predictions ...</p>
  <p>
    <a href="AboutUs.html">About Us</a>
    ...
  </p>
  <p>Copyright © 2014</p>
</footer>
```

在这里，<header>和<footer>元素替代了原来的<div>元素。如果你在修改一个大型网站，可以考虑用HTML5中相应的语义元素来包装已有的<div>元素。

可能你已经注意到了，这个例子里的<header>和<footer>元素仍然使用了类名。这样做的目的就是不用修改原来的样式表。由于类名不变，所以原来应用到<div>元素的样式规则，可以应用给现在的<header>和<footer>元素。

不管怎么说，这里的类名还是有点多余。如果你想把它们都删掉，变成这样：

```
<header>
  <h1>How the World Could End</h1>
  <p class="Teaser">Scenarios that spell the end of life as we know it</p>
  <p class="Byline">by Ray N. Carnation</p>
</header>
```

那就得修改样式表规则，直接通过元素名来应用样式。对于当前页面，这样做没有问题，因为其中只有一个<header>和一个<footer>元素。

下面是修改之后的为<header>及其包含的所有元素应用样式的规则：

```
/*为<header>添加样式，使其具有页眉的外观（蓝色带边框）*/
header {
  ...
}

/*为<header>中的<h1>添加样式（这是文章的标题）*/
header h1 {
  ...
}

/*为<header>中的子标题添加样式*/
header .Teaser {
  ...
}

/*为<header>中的署名行添加样式*/
header .Byline {
  ...
}
```

这两种应用样式的方式会得到相同的结果。正如HTML5中的很多设计问题一样，可能需要充分地讨论，但到底该怎么做，没有硬性规定。

此时，也许你会问：为什么内容部分的<div>元素还保留着呢？这样没有问题，因为HTML5页面经常会混合各种语义元素和更通用的<div>容器。HTML5没有“content”元素，所以仍然可以使用<div>元素。

注意 这个网页在IE9之前的Internet Explorer中无法正确显示。为解决这个问题，可以参考2.3节的有关讨论。不过接下来，还是让我们再多接触几个能够增强页面的语义元素吧。

最后，还有一个元素有必要用在示例页面中。那就是HTML5的<article>元素，这个元素表示一个完整的、自成一体的内容块，比如博客文章或新闻报道。<article>元素应该包含所有相关的内容，包括标题、作者署名以及正文。添加了<article>元素之后的页面结构就变成如下所示：

```
<article>
  <header>
    <h1>How the World Could End</h1>
    ...
  </header>

  <div class="Content">
    <p><span class="LeadIn">Right now</span>, you're probably ...</p>
    <p>...</p>
    <h2>Mayan Doomsday</h2>
    <p>Skeptics suggest ...</p>
    ...
  </div>
</article>
```

```
<footer>
  <p class="Disclaimer">These apocalyptic predictions ...</p>
  ...
</footer>
```

图2-3是最终的页面结构示意图。

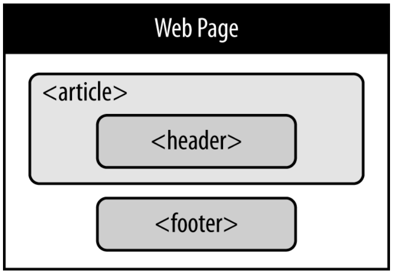


图2-3：重新设计之后，页面中使用了三个新的HTML5语义元素。对于原来的页面结构，我们可以理解为："这是一个包含三个区块的页面。"而对于新的页面结构，可以这样想："这是一篇包含文头和页脚的文章。"

尽管此时的页面在浏览器中看起来还是一样的，但后台已经潜伏了一些额外的信息。此时，如果正好一个搜索机器人造访你的网站，停留在这个页面上，那它就能迅速找到页面的内容（通过`<article>`元素）以及该内容的标题（通过`<header>`元素）。至于页脚，它就不会太在意了。

注意 有时候，需要把一篇文章拆开，分几个页面来显示。目前关于这个问题的处理方案，就是把文章的每一部分都放在它自己的`<article>`元素中——即使内容并不完整，也不是自成一体的。在实际使用语义元素的时候，语义元素与页面表现之间经常会发生冲突，这只是一个典型的例子。

2.2.3 用<figure>添加插图

很多页面中都包含图片。但是，插图（figure）与图片的概念还不完全一样。HTML5规范建议我们把插图想象成一本书中的附图；换句话说，插图虽然独立于文本，但文本中会提到它。

一般来说，插图应该是浮动的；换句话说，应该把它放在相关文本旁边的一个比较近便的位置上，而不要把它们锁定在特定的词或元素旁边。而且，插图通常还会有与之相伴的浮动图题。

下面是给这篇启示录般的文章添加插图的HTML标记。其中也包含正好在它前面和后面的两个段落，这样你就能知道插图在标记中的确切位置了。

```
<p><span class="LeadIn">Right now</span>, you're probably ...</p>
<div class="FloatFigure">

  <img src="human_skull.jpg" alt="Human skull">
```

```
  <p>Will you be the last person standing if one of these apocalyptic
  scenarios plays out?</p>
</div>

<p>But don't get too smug ...</p>
```

光有这些标记还不行，必须还得有相应的样式表规则，才能把插图定位到适当的位置（同时添加外边距、控制图题文本的格式，当然你也可以自己给它添加一个边框）。下面就是本书给出的样式规则：

```
/*为插图应用样式*/
.FloatFigure {
  float: left;
  margin-left: 0px;
  margin-top: 0px;
  margin-right: 20px;
  margin-bottom: 0px;
}

/*为图题应用样式*/
.FloatFigure p {
  max-width: 200px;
  font-size: small;
  font-style: italic;
  margin-bottom: 5px;
}
```

图2-4展示了目前这个示例页面的外观。

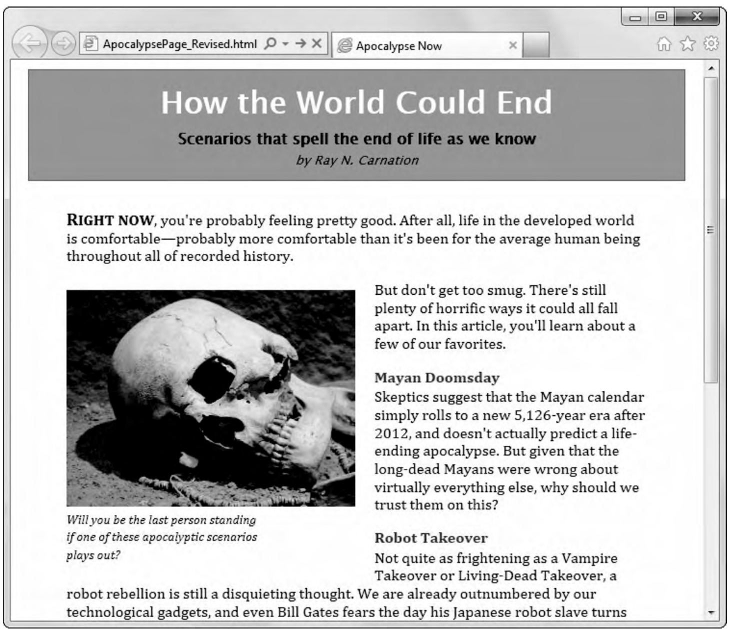


图2-4：现在，我们通过插图美化了文本。在标记中，插图恰好在第一段文本之后，因此它会浮动到后面文本的左侧。注意一下图题文本的宽度，我们通过限制这个宽度，让图题内容显示很充实、很优雅

如果你以前也是这样为内容添加插图的，那在听说HTML5专门为你量身打造了一个新的语义元素，一定会高兴得合不拢嘴。没错，在这里可以不再用那个乏味的<div>了，因为现在有了一个专门的<figure>元素。那么图题呢？图题可以放在<figure>中的<figcaption>元素里：

```
<figure class="FloatFigure">
   <img src="human_skull.jpg" alt="Human skull">
   <figcaption>Will you be the last person standing if one of these
   apocalyptic scenarios plays out?</figcaption>
</figure>
```

当然，给插图和图题应用什么样式，把它们放在什么位置上，还是由你说了算。对这个例子来说，你得修改样式规则的选择符，才能重新选中图题文本。现在使用的是.FloatFigure p，而修改之后应该是.FloatFigure figcaption。

提示 在这个例子中，<figure>元素由于有一个类名（FloatFigure），所以仍然会被应用样式，这与元素名无关。使用类名是因为我们要为不同的插图应用不同的样式。比如，可以让有的插图靠左浮动，而让有的插图靠右浮动，有的插图还要应用不同的外边距或图题格式，等等。为了保留这种灵活性，还是有必要继续通过类来为插图应用样式。

打开浏览器看一下，插图还是那样。但不同的是标注插图的标记现在的含义已经非常明确了。（顺便提个醒，<figcaption>元素不是只能包含文本——任何HTML元素都可以，比如链接、小图标等。）

最后，有必要再说一件事。因为图题中经常就包含了对图片的完整说明，所以alt文本就显得有点多余了。这种情况下，可以把<img>元素中的alt属性删除：

```
<figure class="FloatFigure">
   <img src="human_skull.jpg">
   <figcaption>A human skull lies on the sand</figcaption>
</figure>
```

这里要小心的是，不能把alt文本设成空字符串。因为这就意味着你的图片纯粹是装饰用的，屏幕阅读器会忽略不读。

2.2.4 用<aside>添加附注

新的<aside>元素表示与它周围的文本没有密切关系的内容。这就是说，你可以像在印刷品中使用附注栏一样使用<aside>元素，可以通过它介绍另一个相关的话题，或者对主文档中提出的某个观点进行解释。（比如，可以直接翻到本节末尾的附注栏。）另外，也可以用<aside>来盛放广告、相关内容链接，甚至如图2-5所示的醒目引文（pull quote）。

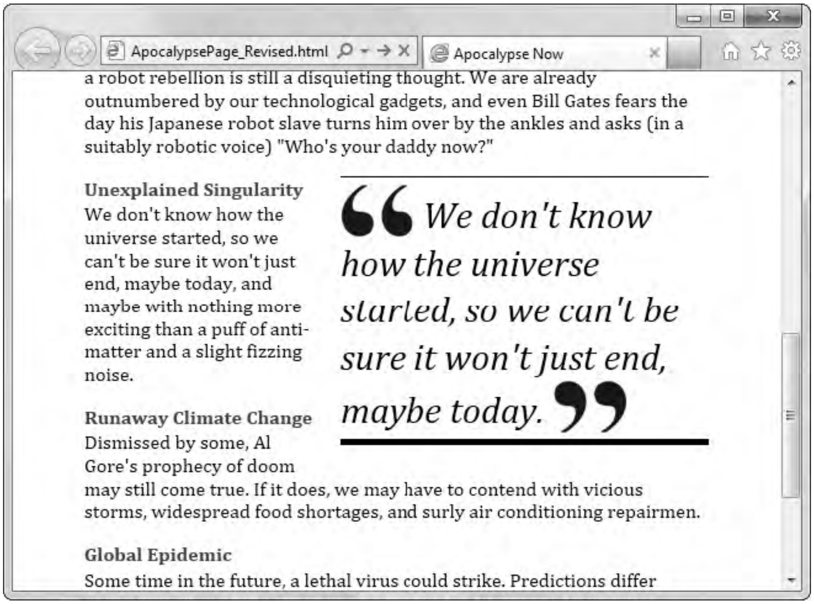


图2-5：醒目引文也是从印刷行业中借用的技术，用于吸引读者注意力，突出重要的内容

当然，使用熟悉的<div>元素也可以创造这种效果，但用<aside>元素包装同样的内容，可以让标记更富有意义：

```
<p>... (in a suitably robotic voice) "Who's your daddy now?"</p>

<aside class="PullQuote">
  <img src="quotes_start.png" alt="Quote">
  We don't know how the universe started, so we can't be sure it won't
  just end, maybe today.
  <img src="quotes_end.png" alt="End quote">
</aside>

<h2>Unexplained Singularity</h2>
```

这一次，样式表把醒目引文浮动到了右边。为满足你的好奇心，下面给出了相应的样式：

```
.PullQuote {
  float: right;
  max-width: 300px;
  border-top: thin black solid;
  border-bottom: thick black solid;
  font-size: 30px;
  line-height: 130%;
  font-style: italic;
  padding-top: 5px;
  padding-bottom: 5px;
  margin-left: 15px;
  margin-bottom: 10px;
}
```

```
.PullQuote img {
  vertical-align: bottom;
}
```

语义元素是怎么来的

在发明HTML5之前，其发明者花了很长时间研究已有的网页。他们不是光浏览自己喜欢看的站点，还研读了谷歌对**十亿个**网页的统计信息。（如果你对这个出色的研究感兴趣，可以看看这里：https://developers.google.com/webmasters/state-of-the-web/。）

谷歌公布的这个调查分析并列出了Web作者在自己网页中使用的类名。谷歌的目的是想告诉大家，你们使用的类名可能与匹配的元素的用途相悖，从而为大家构造网页提供有价值的参考。比如，要是所有人都给一个<div>元素指定名为header的类，那么就可以推断出大家都把页眉内容放在网页的顶部。

谷歌发现的最重要的一个现象，就是大量的网页根本不使用类名（甚至连样式表都没有）。然后，他们汇编了一组最常用的类名，包括footer、header、title、menu、nav，分别对应着HTML5中的<footer>、<header>、<nav>等新语义元素。另外一些由其他人建议但尚未创造出来的语义元素有search和copyright等。

换句话说，Web页面拥有一些共性的东西，比如页面都有页眉、页脚、侧边栏和导航菜单。只不过大家在区分这些构造时采用的方式多多少少有点不一致。基于这个认识，从人们普遍的做法中提取出相应的语义，据以为HTML语言添加一些新元素，也就成了自然而然的事了。这就是HTML5中语义元素的来历。

2.3 浏览器对语义元素的支持情况

到目前为止，我们做练习做得还是很开心的。不过，这个结构良好的页面如果在旧版本浏览器里没法看，那用处可就不大了。

所幸的是，HTML5的这些语义元素已经基本得到了所有现代浏览器的支持。就算你想找到一个不支持它们的Chrome、Firefox、Safari或Opera的版本，都不太可能。最大的绊脚石还是IE9之前的Internet Explorer，包括仍然占很大比例的IE8。

幸运的是，语义元素还是一个比较容易弥补的功能。毕竟，语义元素本身什么也不做，要支持它们，只要让浏览器把它们当做普通的<div>元素就行了。为此，我们要做的就是像接下来几节所说的，为它们添加点样式规则。之后就可以得到超级可靠的语义元素，即使用10年前的老古董浏览器来看，也不是问题了。

注意 如果你使用了Modernizr（见1.6.7节），那你的页面会自动完成语义元素的修补工作。那么接下来几节的内容你就可以不看了。如果你没有使用Modernizr，或者你很想知道怎么完成修补工作，那请继续阅读。

2.3.1 为语义元素添加样式

浏览器在遇到不认识的元素时，会把它们当成内联（inline）元素。大多数HTML5语义元素（包括本章已经介绍的这些，但除<time>之外）都是块级元素，也就是需要在单独一行上来呈现它们，同时在它们与前后元素之间各添加一些空间。

不认识HTML5语义元素的浏览器不知道应该把它们显示为块级元素，所以很可能会把它们都挤到一起。为解决这个问题，只要在样式表中添加一条“超级规则”即可。下面就是一条为9个HTML5元素应用块级显示格式的规则：

```
article, aside, figure, figcaption, footer, header, main, nav, section,
summary {
  display: block;
}
```

这条规则对于能够识别HTML5元素的浏览器没有作用，因为它们的display属性已经被默认设成了block。而且这条规则也不影响我们已经为这些元素应用的样式。那些样式照样可以添加到它们身上。

2.3.2 使用HTML5“垫片”

对于大多数浏览器而言，上一节介绍的技术可以解决兼容问题。但这里的“大多数”并不包括IE8及更早的版本。换句话说，对于较早版本的IE，还要面临一个挑战：它们会拒绝给无法识别的元素应用样式。好在，有一个变通方案：通过JavaScript创建新元素，就可以骗过IE，让它识别外来元素。比如，下面的脚本可以让IE识别并为<header>元素应用样式：

```
<script>
  document.createElement('header')
</script>
```

实际上，你不用自己亲手写这些代码，因为已经有人为你写好了，只要拿过来用就可以了。要使用这个脚本，只要在页面的<head>区块中引用它即可，就像这样：

```
<head>
  <title>...</title>
  <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
  ...
<head>
```

这行代码会从html5shim.googlecode.com这个Web服务器上取得脚本，然后在浏览器处理页面其余部分之前运行。这个脚本使用前面描述的JavaScript代码创建所有的新HTML5元素，然后还会为它们动态应用上一节提到的样式，确保新元素能正确地显示为块元素。现在，剩下的事儿就是使用这些元素，并为它们应用你自己的样式。

对了，刚才忘说了，前面的html5.js脚本应该是有条件执行的——只在你使用旧版本Internet Explorer的情况下才会执行。为了避免不必要地加载JavaScript文件，可以像下面这样把引用脚本的代码放在IE的条件注释中：

```
<!--[if lt IE 9]>
  <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

这样，其他浏览器（IE9及更高版本）就会忽略这行脚本，为你的页面节省数毫秒的加载时间。

提示 前面的例子直接使用了来自谷歌代码托管站点的垫片脚本。你也可以把它下载下来，地址为http://tinyurl.com/the-shiv。下载后放到本地目录里，改一下页面中引用这个脚本的地址即可。最后，还要提醒你，如果你是在本地计算机上测试网页（而不是上传到服务器），Internet Explorer会自动将页面置为受限模式。意思就是，你会在页面顶部看到臭名昭著的IE安全提示条，告诉你Internet Explorer已经禁用了其中的所有脚本，包括HTML5垫片脚本。要想正常测试，必须单击该安全提示条，选择允许活动的内容。

如果你把页面上传到网站中，就没有这个问题了，不过这无疑会增加测试的工作量。解决方案呢？就是在页面开头添加“Web标志”，详见1.3.5节。

2.3.3 Modernizr：一站式解决方案

要解决为语义元素应用样式的问题，还有一个方案：使用Modernizr（参见1.6.7节）。Modernizr内置了HTML5垫片脚本，因此会自动替你解决上述问题，不用你再使用html5.js或者添加什么样式规则了。因此，如果你已经在用Modernizr检测功能了，那么要考虑的就只有怎么使用语义元素了。

2.4 使用语义元素设计站点

在一个简单、类似文档的页面中应用语义元素相对容易。如果是把它们应用到整个站点也没有任何困难，但必须面对一系列新问题。因为HTML5还是一块未被开垦的处女地，很少有约定俗成的惯例（但却有一大堆合情合理的争议）。这样说吧，假设你准备从两种标记方案中选择一种，HTML5标准会说它们都是完全可以接受的，至于哪个方案最适合你的内容，你自己说了算。

图2-6展示了我们接下来将要设计的一个更有创造性的作品。



图2-6：这里，之前仅有一页的文章页面已经变成了一个完整的基于内容的站点。站点的页眉横跨上方，内容位居其下，而左侧的导航条提供了导航控件、About Us和一张图片广告

2.4.1　理解<header>

对于<header>元素来说，有两种使用方式，但差别并不大。一种是用它标注内容的标题，另一种是用它标注网页的页眉。有时候，这两种用途是重叠的（比如图2-1中那个单页文章的例子）。但有时候，你的网页里不仅要用<header>标注页眉，还要用它去标注很多内容的标题。图2-6所示的例子就是这种情况。

但有时候到底该不该用<header>并不十分明确，因为要标注的内容区块的角色变化会影响最终的决策。如果你处理的是内容，那么除非必要，一般不必使用<header>。只有在内容标题还附带了其他信息的情况下，才有必要考虑<header>。也就是说，其中包含标题、概要、发表日期、作者署名、图片或子主题链接等很多内容，例如：

```
<header>
  <h1>How the World Could End</h1>
  <p class="Tagline">Scenarios that spell the end of life as we know it</p>
  <p class="Byline">by Ray N. Carnation</p>
</header>
```

不过，在为网站创建页眉的时候，人们多数会直接考虑<header>元素，即便这个页眉就是一个CSS控制的长盒子，里面啥都没有。毕竟页眉是网站的一个核心组件，谁知道哪天你要往里面塞点什么东西呢！

结论如下：网页中可以包含多个<header>元素（通常也应该如此），即使相应的区块在页面中的角色不一样。

把网页变成网站

图2-6展示了一个虚构的网站中的一个页面。

在真实的网站中，可能**几十个**甚至更多个页面都会有相同的布局（以及相同的侧边栏）。访客点击页面的链接后，唯一变化的地方就是主页面中的内容——也就是这里的文章。

HTML5并没有什么魔法，它不会让你的网页自动变成网站。换句话说，你还要靠如下所示的以前开发网站的技术和技巧。

- **服务器端框架**。其实背后的思想都很简单：在浏览器请求某个页面时，Web服务器临时将页面的各个部分组装起来，包括公共的元素（如导航条）和内容。这是目前最常用的一种技术，也是构建大型、专业网站的必由之路。以不同方式实现这种手段的技术不计其数，包括很早就出现的服务器端包含功能，一些富Web应用平台（如ASP.NET和PHP），还有内容管理系统（如Drupal和WordPress）。
- **页面模板**。Dreamweaver、Expression Web等功能强大的网页编辑器中就有页面模板功能。模板就是一个定义了页面结构并且包含页面中会重复出现的内容（像页眉和侧边栏）的页面。有了模板，就可以利用它来创建网站的所有页面。最关键的是，更新模板之后，网站编辑器会自动更新使用该模板的所有页面。

图2-6所示的启示录网站用一个<header>元素作为站点的页眉，用另一个<header>元素作为文档的标题区。作为页眉的<header>中有一张横幅图片，图片中的文本也是由图片编辑器生成的：

```
<header class="SiteHeader">
  <img src="site_logo.png" alt="Apocalypse Today">
  <h1 style="display:none">Apocalypse Today</h1>
</header>
```

在此，我们注意到页眉中还包含一个我们在页面中看不到的<h1>元素，其中的内容与图片中的文本内容相同。而一条内联样式隐藏了这个标题。

这个例子提出了一个明显的问题——为什么要添加一个你看不见的标题？原因其实不止一个。首先，<header>元素中都应该包含某个级别的标题，这样做仅仅是为了遵守HTML5的规定。其次，这样设计可以让使用屏幕阅读器的人能够无障碍地阅读，因为他们经常会从一个标题跳到另一个标题，而不会关注标题之间的内容。最后，这样就为页面建立了标题结构，而页面其他部分的标题都要依序选用。如果我们说，在站点的页眉中使用了<h1>元素后，你就应该给页面中其他的区块（如“Articles”和“About Us”）选用<h2>元素作标题，你可能觉得有点怪。要了解为什么这样设计，请参考下面的附注栏“站点的标题结构”。

注意 当然，简单地给页眉中添加一行文字比较省事。（而如果你喜欢一些新奇的字体，还可以考虑CSS3新的Web字体功能，详见6.4节。）但对于诸多把标题放在图片中的网页而言，隐藏标题同样也是个不错的办法。

站点的标题结构

一个页面中可以有多个一级标题吗？这样做到底好不好？

根据HTML的官方说法，一个页面中可以有任意多个一级标题。但是，很多网站开发人员更倾向于每个页面只使用一个一级标题，因为这样能保证网页的无障碍性。（因为使用屏幕阅读器的人在从一个二级标题跳到另一个二级标题的时候，有可能错过中间的一级标题。）也有不少网络维护人员认为，每个页面就应该只有一个一级标题，这个一级标题在整个网站中是独一无二的，可以明确地告诉搜索引擎网站中有什么内容。

图2-6中的例子采用的就是这种风格。站点顶部的标题“Apocalypse Today”是页面中唯一的`<h1>`元素。页面中其他的部分，如侧边栏中的Articles和About Us，使用的都是二级标题。文章的标题使用的也是二级标题。（额外做一点规划，可以让一级标题中也包含当前文章的信息——毕竟，这个标题是不可见的，如此一来可以让通过谷歌等搜索引擎搜索特定关键词的人也能看到这个页面。）

不过，除此之外还有一种风格，也同样是允许的。换句话说，可以在页面中每个主要部分都使用一级元素，比如侧边栏、文章等。

或者，也可以给网站一个一级标题，而把侧边栏的内容用二级标题标注（就跟眼下这个例子一样），但是给文章再应用一个一级标题。在HTML5中，这样没有问题，因为它的纲要系统允许这样。稍后在2.5.3节将会介绍，有些元素（包括`<article>`），是被当作一个独立的区块来处理的，独立的区块可以有自己不同的纲要。这样，诸如此类的独立区块完全可以重新建立标题级别，再从`<h1>`开始。（不过，HTML5也说从任何标题级别开始都挺好的。）

简言之，如何设计站点的标题结构是一个没有唯一答案的命题。不过，随着HTML5获得胜利并统治Web，好像多个`<h1>`的设计会越来越时髦。不过现在，很多开发人员为了让屏幕阅读器开心，仍然坚持只使用一个`<h1>`。

2.4.2 用<nav>标注导航链接

这个启示录网站中最有意思的新功能就是左侧的侧边栏，其中包含网站的导航、其他信息和一张图片广告。（一般来说，广告位都是放一段JavaScript脚本，从类似Google AdSense等服务随机地提取广告。但我们这个例子就硬编码了一张图片，就算是一种替代吧。）

在传统的HTML网站中，你可能会把整个侧边栏都放到一个`<div>`中。而在HTML5时代，则应该主要使用两个针对性更强的元素：`<aside>`和`<nav>`。

要说<aside>元素，倒是跟<header>元素有点像，哪里像呢？<aside>的含义也有一点细微的发散。可以用它来标注一段与正文相关的内容（2.2.5节正是这么做的），也可以用它表示页面中一个完全独立的区块——作为页面主要内容的陪衬。

而<nav>元素则用于包装一组链接。这些链接可以指向当前页面中的主题，也可以指向网站中的其他页面。多数页面中都会包含多个<nav>区块。但并不是所有链接都需要<nav>区块——相反，<nav>通常只用于页面中最大最主要的导航区。例如，对于一组文章的链接（像图2-6那样的），绝对需要一个<nav>区块。可是，如果只是放在页面底部几个许可或联系信息的链接，那大可不必麻烦<nav>。

明白了这两个元素的用途，接下来该付诸实践了。首先，再看一看图2-6中的侧边栏。然后，在一张纸上画一画，看你打算怎么用标记体现其中内容的结构。最后，再接着往下看，一块来找一个最佳方案。

事实上，如图2-7所示，至少有两个比较合理的方式来构造侧边栏。

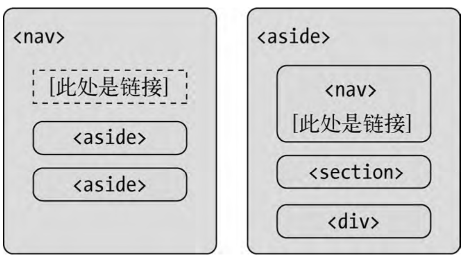


图2-7：左：可以把整个侧边栏想象成一个导航区，在其中填入一些内容。这样，整个侧边栏可以放在一个<nav>中，其他内容可以使用<aside>（因为其中的内容与侧边栏的主要内容——链接无关）。右：或者，可以把整个侧边栏想象成一个独立的多用途的网页区块。这样，侧边栏就是一个<aside>，而其中有关导航的内容放在<nav>里面

这个启示录站点使用的是第二种方案（即图2-7中右侧的结构）。选择该方案主要是因为这里的侧边栏有多种用途，而没有哪一种用途是主要用途。不过，要是这里的导航又长又复杂（比如都用上了可折叠的菜单形式），只是后面跟着一小段内容，那第一种方案就更合适一些。

以下就是构成侧边栏的标记，包含三个区块：

```
<aside class="NavSidebar">
  <nav>
    <h2>Articles</h2>
    <ul>
      <li><a href="...">How The World Could End</a></li>
      <li><a href="...">Would Aliens Enslave or Eradicate Us?</a></li>
      ...
    </ul>
  </nav>

  <section>
    <h2>About Us</h2>
    <p>Apocalypse Today is a world leader in conspiracy theories ..."
    </p>
  </section>
```

```
  <div>
    <img src="ad.jpg" alt="Luckies cigarette ad: it's toasted">
  </div>
</aside>
```

看完这些代码，应该注意到如下关键点。

- **标题（Articles和About Us）使用的是二级标题**。这样，它们就自然位于网站的一级标题之下，从而方便屏幕阅读器无障碍地阅读标题。
- **链接以<ul>和<li>元素组成的无序列表来标记**。网站设计师普遍认为，处理一组链接的最佳方式，也是最无障碍的方式，就是使用列表。不过，你可能得通过样式表删除列表项默认的缩进（这个例子已经这样做了）和项目符号（本例没有项目符号）。
- **“About Us”包含在一个<section>元素中**。这是因为没有合适的语义元素。<section>当然要比<div>更具体一点，它适合任何以标题开头的内容区块。假如有一个更具体的元素（例如，假设有一个<about>元素），那么这里就不会用<section>，可惜没有。
- **图片广告放在一个<div>里**。如前所述，<section>元素只适合带标题的内容，而这里的图片区块没有标题。（而如果真有的话，比如“A Word from Our Sponsors”，那就该用<section>元素了。）从技术角度看，根本没有必要把图片还放在其他元素中。但有了<div>，区块之间的关系就更明确了，这样更容易分别为不同的区块应用样式，或者在必要时通过JavaScript分别操作它们。

实际上，还有一些细节，这个侧边栏里没有，但可能其他侧边栏里会有。比如，复杂的侧边栏可能会以<header>和<footer>作为开头和结尾，也有可能包含多个<nav>区块——存档文章链接一个、新闻报道链接一个、友情链接或相关站点链接一个，等等。有兴趣的话，读者可以自己找找一些有代表性的博客，其中的侧边栏往往包含很多区块，其中不少都是导航用的。

这里为<aside>标注的侧边栏应用样式的规则，与为传统的<div>标注的侧边栏应用样式的规则一样。它们都会把侧边栏摆放到正确的位置上，使用绝对定位，设置某些格式上的细节，如内边距、背景，等等：

```
aside.NavSidebar
{
  position: absolute;
  top: 179px;
  left: 0px;
  padding: 5px 15px 0px 15px;
  width: 203px;
  min-height: 1500px;
  background-color:#eee;
  font-size: small;
}
```

这条规则后面是一些上下文样式表规则，分别为侧边栏中的<h2>、<ul>、<li>以及<img>元素应用样式。（跟以前一样，可以从http://prosetech.com/html5下载示例代码，然后仔细研究一下其中的所有样式规则。）

注意 我们已经介绍了，<nav>通常会单独出现，有时候也会出现在<aside>中。其实，还有一个地方也经常可以看到它的身影：网页顶部的<header>元素中。

理解了这个侧边栏是如何构造的，也就容易理解它与整个页面布局的关系了，如图2-8所示。

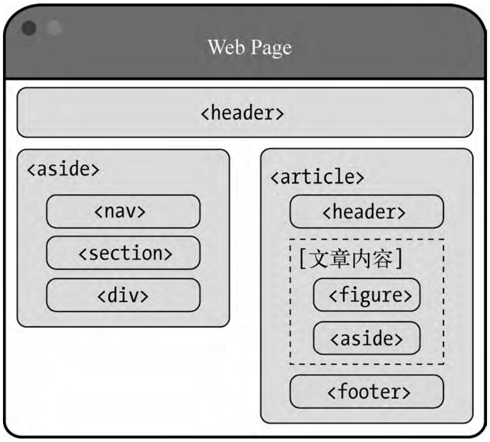


图2-8：这里展示了启示录网页（图2-6）中用到的所有语义元素

使用<details>和<summary>的折叠框

你肯定在有的网页上见过可以折叠的区块：通过单击区块标题能够显示或隐藏其中的内容。折叠框是用JavaScript能够轻易实现的众多界面元素之一。只要在单击标题的时候，适当改变样式设置，隐藏内容：

```
var box = document.getElementById("myBox");
box.style.display = "none";
```

或者再把内容显示出来即可：

```
var box = document.getElementById("myBox");
box.style.display = "block";
```

有意思的是，HTML5还新添了两个语义元素，用于辅助将这种行为自动化。具体做法是把可以折叠的区块放在一个<details>元素中，而把标题放在一个<summary>元素中。最终结果类似如下这样：

```
<details>
  <summary>Section #1</summary>
  <p>If you can see this content, the section is expanded</p>
</details>
```

支持这两个元素的浏览器（目前只有Chrome支持）只会显示标题——可能还会带有视觉上的提示（比如在标题旁边放在一个小三角形图标）。然后，用户单击标题，再把完整的内容

显示出来。不支持<details>和<summary>的浏览器则上来就会显示所有内容，用户也不能把内容折叠起来。

目前，业内对<details>和<summary>元素还有争议。很多Web开发人员认为它们并不是真正的语义元素，因为它们更倾向于视觉表现，而非逻辑结构。

我们的建议是最好不要使用<details>和<summary>元素，因为支持它们的浏览器太少了。虽然可以用JavaScript写一段脚本在不支持它们的浏览器中使用，但编写这种脚本不如你自己实现一个在任何浏览器中都可以折叠的方案，毕竟那样只要几行代码而已。

2.4.3 理解区块

如前所述，区块元素<section>是应该最后考虑的语义元素。如果有一个带标题的内容块，而其他语义元素都不合适，那么选择<section>通常比选择<div>更好一些。

那么通常应该在<section>元素中放什么呢？这跟你的看法有关，它可能是一个能够适合各种需求的灵活的工具，也可能是一个松松垮垮没有明确身份的怪物。之所以会有这么大的区别，主要是因为<section>在网页中可以扮演很多不同的角色。可以放在<section>中的内容有以下几种。

- 与页面主体内容并列显示的小内容块，例如我们启示录网站中的About Us段落。
- 独立性内容，但却不能用文章（<article>）来描述，比如客户的购物记录或产品清单。
- 分组内容——例如，新闻站点中的一组文章。
- 比较长的文档中的一部分。比如，在启示录网站的文章中，可以用它把每种世界末日的情形标注为一个独立的区块。有时候，这样使用区块是为了保证文档能有一个正确的纲要，而这正是下一节我们要介绍的内容。

前面列出的最后两项可能会令你觉得很不可思议。很多Web开发人员会觉得用一个元素既可以标注一篇文章中的一个片段，又可以标注整个一组文章，这伸缩性似乎有点大了。有人认为HTML5至少应该用两个不同的元素来对应这两种情况。但HTML5的创造者希望一方面保持简单（限制新元素的数量），另一方面也让新元素尽可能灵活且实用。

最后，还有一件事儿要考虑。<section>元素也会影响网页的纲要，也就是2.5节要讨论的主题。

2.4.4 理解<footer>

HTML5与内容丰富的“胖”<header>可谓天生一对。在<header>中不仅可以放副标题和作者署名，还可以添加图片、导航区块（使用<nav>元素），以及任何有必要放在页面顶部的内容。

奇怪的是，HTML5对<footer>元素可就不那么有人情味了。HTML5规定，只能在<footer>元素中放一些网站版权信息、作品来源、法律限制及链接之类的信息。不能在<footer>里面放太多链接、重要的内容或无关的内容，如广告、社交媒体按钮以及网站部件等。

这就提出了一个问题：如果你的网站需要一个“胖”<footer>怎么办？毕竟，“胖”页脚目前非常流行（见图2-9中的例子）。这些“胖”页脚经常会用到如下所示的一些花哨的技术。

- **固定定位**，这样就可以让页脚始终固定在浏览器窗口底部，无论访客如何滚动都无所谓（如图2-9中的例子所示）。
- **关闭按钮**，这样用户在看完页脚内容后，单击它们就可以腾出页面空间（如图2-9中的例子所示）。为实现这个功能，要使用一点JavaScript代码（与前一节附注栏中使用的代码相似），以便隐藏包含页脚的元素。
- **部分透明的背景**，这样就可以透过页脚看到内容。如果页脚正在宣布即时新闻或重要的免责声明，部分透明的背景很合适，通常与关闭按钮联用。
- **动画**，页脚在视图中弹出，或者滑入视图（例如，可以看看在http://www.nytimes.com上阅读到一篇文章最后时弹出的相关文章提示框）。

如果你的站点中包含这种页脚，就要作出选择。比较简单的办法是无视规定。这个办法并没有听起来那么可怕，因为其他网站的开发人员也在犯同样的错误。而随着时间推移，官方的规定也可能会放开，允许用<footer>包含奇特的页脚。可是，假如你现在不想违反标准的规定，那就得调整一下标记。好在调整标记不难。



图2-9：这个荒唐的“胖”页脚中包含不少多余的内容，比如一个奖项图片和一些社交媒体按钮。它使用了固定定位使自己一直显示在浏览器窗口底部，就像一个工具条。好在这个页脚有一个地方可以弥补它的缺点，那就是右上角的关闭按钮，单击该按钮就能让这个页脚从视野中消失

调整标记的关键在于从多余的内容中提取出标准的页脚来。在浏览器中，这些内容看起来还是一个页脚，但在标记中，其他内容都不包含在<footer>元素中。例如，以下就是图2-9中“胖”页脚的实际结构：

```
<div id="FatFooter">
  <!–“胖”页脚的内容 -->
  <img onclick="CloseBox()" src="close_icon.png" class="CloseButton">
  ...
  <footer>
    <!-- 标准页脚的内容 -->
    <p>The views expressed on this site do not ... </p>
```

```
  </footer>
</div>
```

最外围的<div>没有特殊含义，它只是负责把多余的内容和标准的页脚内容包装起来。同样，可以给它应用一些样式表规则，以便将其锁定在合适的位置上：

```
#FatFooter {
  position: fixed;
  bottom: 0px;
  height: 145px;
  width: 100%;
  background: #ECD672;
  border-top: thin solid black;
  font-size: small;
}
```

注意 在这个例子中，样式表规则是通过ID应用样式的（使用#FatFooter选择符），并没有使用类名（如.FatFooter选择符）。这是因为“胖”页脚本身已经有了唯一的ID，目的是为了方便JavaScript代码在用户单击关闭按钮时可以找到并隐藏它。这种情况下，在样式表中使用唯一的ID要比另外添加一个类名更可取。

当然，也可以把页脚中的其余内容放在一个<aside>元素中，从而清楚地表明其中的内容属于一个独立的区块，与页面中的其他内容无关。相应的标记结构类似如下所示：

```
<div id="FatFooter">
  <aside>
    <!–“胖”页脚的内容 -->
    <img onclick="CloseBox()" src="close_icon.png" class="CloseButton">
    ...
  </aside>

  <footer>
    <!-- 标准页脚的内容 -->
    <p>The views expressed on this site do not ... </p>
  </footer>
</div>
```

这里最重要的一点是没有把<footer>放到<aside>元素中。因为<footer>并不属于<aside>，而是整个网站的一部分。类似地，如果有一个<footer>属于某些内容，那么就应该将这个<footer>放在包含相应内容的元素中。

注意 正确使用HTML5语义元素的规则和指导方针还在变化。在HTML社区中，有关如何在大型、复杂站点中正确使用标记的话题，常常会引起激烈的争论。在此，我给大家一个建议：如果某个元素看起来不适合标注你的内容，那就不要用。当然，你也可以上网去讨论，很多超聪明的HTML大师会为你现身说法。（最好的一个去处就是http://html5doctor.com，在该网站大多数文章的评论中，都可以看到正在争论中的话题。）

2.4.5 使用<main>标识主要内容

HTML5还有一个经常被人忽视的<main>元素，用于标识网页的主要内容。比如，在前面的启示录网站上，主要内容就是整篇文章，不包含站点页眉、页脚和侧边栏。强烈建议大家在自己的网页中使用这个元素。

在启示录网站中，我们就用<main>包装了<article>：

```
<!DOCTYPE html>
<html lang="en">
<head>
  ...
</head>

<body>
  <header>
      ...
  </header>
  <aside>
      ...
  </aside>

  <main>
    <article>
      ...
    </article>
  </main>
  <footer>
    ...
  </footer>
</body>
</html>
```

不能把<main>嵌套到<article>（或其他任何语义元素）里边。这是因为<main>元素的使命就是包裹页面中的主要内容，而非标识文档中某个重要的部分。换句话说，一个页面中只能有一个<main>元素。

乍一看好像<main>也没有大用处。可是它对屏幕阅读器很重要，有了它，屏幕阅读器就可以跳过那些次要的内容，比如页眉、导航菜单、广告和侧边栏，直达内容。我们这个例子中<main>紧挨着<article>，这不是必须的。在复杂些的页面中，比如包含多个<article>元素时，<main>就应该包含所有<article>，像这样：

```
<main>
  <article>
    ...
  </article>

  <article>
    ...
  </article>

  <article>
```

```
    ...
  </article>

  ...
</main>
```

很明显，每个<article>都包含自己的内容，而<main>包含所有文章。

就算页面中没有<article>元素，同样可以使用<main>元素。比如，页游或应用页面可以用<main>元素包裹用于创建游戏和应用的标记。总之一句话，就是<main>只应该包含主要内容，而不是上下左右的外部细节。

> **注意** <main>是一个相对新的元素，是在HTML5的升级版HTML 5.1（参见前言中的“什么时候可以使用HTML5”）中引入的。

2.5 HTML5 纲要

HTML5定义了一组规则，用于说明如何为网页创建文档纲要（document outline）。网页的纲要能够提供很多便利。例如，浏览器可以让你从纲要中的一处跳到另一处。设计工具可以让你通过在纲要视图中拖放来重排区块。搜索引擎可以使用纲要构建更好的页面预览，而屏幕阅读器获得的便利最大，通过使用纲要可以引导视力有障碍的用户在深度嵌套的区块和子区块中导航。

不过，这些情形还没有一项成为现实，因为除了下一节将要介绍的几个开发人员工具，今天几乎还没有谁在使用HTML5纲要。

> **注意** 一个不影响页面在浏览器中的表现，也没有其他工具使用的功能很难让人兴奋。但是，通过评审网页（至少是网站中典型网页）的纲要来确保其结构合理，而且你也没有破坏HTML5的规则，这个想法还是很不错的。

2.5.1 如何查看纲要

要真正理解纲要，只需看一看你的网页生成的纲要是什么样子的。目前，还没有浏览器实现HTML5纲要（或者给你提供一种查看方式）。不过，倒是有几个工具填补了这个空白。

- **在线HTML纲要生成器**。访问http://gsnedders.html5.org/outliner/，告诉纲要生成器你想为哪个网页生成纲要。如同我们在1.4.2节使用的HTML5验证器一样，可以通过三种方式提交网页：从本地电脑中上传、提供URL或直接在文本框粘贴标记。
- **Chrome扩展**。在Chrome浏览器中查看网页纲要时，可以使用h5o插件分析纲要。访问http://code.google.com/p/h5o安装该插件，然后在网上找开一个HTML5页面看一看（可惜的是，在写作本书时，h5o不能分析本地计算机中的文件）。然后浏览器地址栏中会出现一个纲要图标，单击该图标就会显示页面的结构（如图2-10所示）。h5o的页面中也提供了

一个书签工具（bookmarklet）——可以添加到浏览器书签列表中的一小段JavaScript代码，通过它可以在Firefox和Internet Explorer中显示页面的纲要，但也时候也会出点小状况。

- **Opera扩展**。Chrome的h5o扩展也有一个针对Opera开发的版本，安装地址在http://tinyurl.com/3k3ecdy。

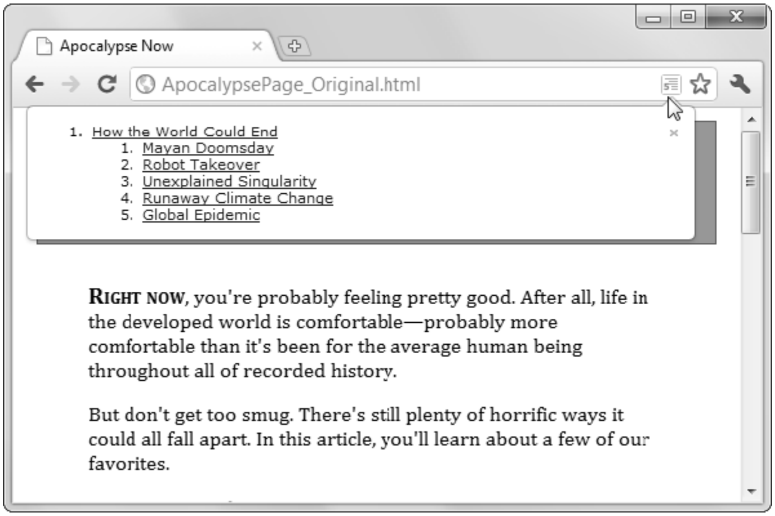


图2-10：在通过安装了h5o的Chrome浏览器访问HTML5网页时，地址栏中会出现一个纲要图标，单击它就会弹出一个包含整个页面纲要的小窗口

2.5.2　基本纲要

要知道自己网页的纲要是什么样子，可以想象把页面中的所有内容都剥离，只剩下编号的标题元素（<h1>、<h2>、<h3>等）中的文本。然后，根据它们在标记中的位置缩进标题，那么嵌套最深的标题在纲要中的缩进也最多。

以最初的还没有应用HTML5元素之前的启示录文章页面的标记为例：

```
<body>
  <div class="Header">
    <h1>How the World Could End</h1>
    ...
  </div>
  ...
  <h2>Mayan Doomsday</h2>
  ...
  <h2>Robot Takeover</h2>
  ...
  <h2>Unexplained Singularity</h2>
  ...
  <h2>Runaway Climate Change</h2>
  ...
  <h2>Global Epidemic</h2>
  ...
```

```
  <div class="Footer">
    ...
  </div>
</body>
```

这个简单的结构会生成如下所示的纲要：

1. How the World Could End
    1. Mayan Doomsday
    2. Robot Takeover
    3. Unexplained Singularity
    4. Runaway Climate Change
    5. Global Epidemic

两个级别的标题（<h1>和<h2>）就创建两级的纲要。这种对应关系与字处理软件中的纲要功能类似。比如，可以在微软Word 的文档结构图窗格中看到类似的纲要。

换一个例子，对于下面的标记：

```
<h1>Level-1 Heading</h1>
<h2>Level-2 Heading</h2>
<h2>Level-2 Heading</h2>
<h3>Level-3 Heading</h3>
<h2>Level-2 Heading</h2>
```

会生成如下所示的纲要：

1. 一级标题
    1. 二级标题
    2. 二级标题
        1. 三级标题
    3. 二级标题

同样，没有什么意外。

最后，纲要算法也很聪明，能够自动忽略跳过的级别。例如，如果你写的标记有点不太规范，从<h1>直接就跳到了<h3>：

```
<h1>Level-1 Heading</h1>
<h2>Level-2 Heading</h2>
<h1>Level-1 Heading</h1>
<h3>Level-3 Heading</h3>
<h2>Level-2 Heading</h2>
```

那会得到如下纲要：

1. 一级标题
    1. 二级标题
2. **一级标题**
    1. **三级标题**
    2. 二级标题

这样，根据其文本中的位置，本来的三级标题与二级标题放在了同一层次上。表面上看，这与浏览器喜欢干的自动纠错类似，但实际上这样处理有特殊的原因。在某些情况下，网页可能是由很多独立的部分拼起来的——例如，可能会包含在其他站点上发布的一篇文章。此时，嵌入内容的标题级别很可能与网页其他部分不会完美地匹配。但由于纲要算法可以消除这些差异，这种问题自然也就不是问题了。

2.5.3 分块元素

分块元素（sectioning element）是指那些在页面中创建新的、嵌套纲要的元素。这些元素有`<article>`、`<aside>`、`<nav>`和`<section>`。要理解分块元素的作用，可以想象一个包含两个`<article>`元素的页面。因为`<article>`是一个分块元素，所以这个页面中（至少）有3个纲要：整个页面有一个纲要、每篇文章有一个嵌套的纲要。

为了进一步弄清楚到底是什么状况，我们来看一看用HTML5改造之后的启示录文章页面：

```
<body>
  <article>
    <header>
      <h1>How the World Could End</h1>
      ...
    </header>

    <div class="Content">
      ...
      <h2>Mayan Doomsday</h2>
      ...
      <h2>Robot Takeover</h2>
      ...
      <h2>Unexplained Singularity</h2>
      ...
      <h2>Runaway Climate Change</h2>
      ...
      <h2>Global Epidemic</h2>
      ...
    </div>
  </article>

  <footer>
    ...
  </footer>
</body>
```

把这些标记复制到http://gsnedders.html5.org/outliner中，可以看到下面的结果：

1. *Untitled Section*
    1. How the World Could End
        1. Mayan Doomsday
        2. Robot Takeover
        3. Unexplained Singularity

4. Runaway Climate Change
5. Global Epidemic

这个纲要开始于一个无标题的区块（Untitled Section），也就是最上层的<body>元素。然后，<article>元素开始一个新的嵌套的纲要，其中包含一个<h1>元素和几个<h2>元素。

有时候，Untitled Section意味着一个错误。尽管<aside>和<nav>元素可以不带标题，但<article>和<section>则万万不可。拿前面的例子来说，无标题的区块是页面中的主区块，属于<body>元素。因为页面只包含一个<article>，所以页面本身没有必要再单独带一个标题，出现这个Untitled Section也没有办法，你可以忽略它。

现在，再来看一个复杂一点的例子，比如带有导航侧边栏的启示录站点（参见2.4节）。把相应的标记放到纲要生成器中，可以得到如下结果：

1. *Apocalypse Today*
    1. *Untitled Section*
        1. Articles
        2. About Us
    2. How the World Could End
        1. Mayan Doomsday
        2. Robot Takeover
        3. *Untitled Section*
        4. Unexplained Singularity
        5. Runaway Climate Change
        6. Global Epidemic

这一次，标记中包含两个分块元素，因此就有两个嵌套的纲要：一个属于侧边栏，另一个属于文章。此外，也有两个无标题区块，但都是合乎规范的——第一个是侧边栏的<aside>元素，第二个是文章中醒目引文的<aside>元素。

注意 除了分块元素之外，还有一些元素被称作**区块根**（section root）。这些元素不是从已有纲要向下分支，而是产生自己的纲要，但不会出现在包含页面的主纲要视图中。比如包含网页内容的<body>元素就是一个区块根，这当然合理。但HTML5还把下列元素视为区块根：<blockquote>、<td>、<fieldset>、<figure>和<details>。

分块元素对复杂页面的意义

分块对于**联合**（syndication）和**聚合**（aggregation）都有很大意义，这两种方式都是从一个页面中取得内容，然后注入到另一个页面的技术。

例如，设想有一个网页，包含一些文章的节选内容，而这些内容都来自另外一个网站。假设这个网页本身有很深的标题嵌套结构，而在这个嵌套结构的某个地方——比如在一个<h4>

标题下面——有一篇从其他网站抓来的文章。

在传统的HTML中，我们希望抓来的文章中的一级标题使用<h5>元素，因为它被嵌套在<h4>下方。可是，这篇文章的结构本来是针对其他网站设计的，而原来的页面没有什么嵌套，因此其标题很可能从<h2>甚至从<h1>开始。虽然这不影响页面显示，但却打乱了标题层级，特别是对屏幕阅读器、搜索引擎以及其他页面处理工具，则会造成更大的困难。

而在HTML5中，这个页面根本不是问题。只要你把这篇文章嵌套在一个<article>元素中，那么被抓过来的内容就会变成它自己嵌套纲要的一部分。而这个纲要可以从任意级别的标题开始，几级标题都无所谓。真正重要的是标题在包含文档中的位置。因此，如果这个<article>元素排在<h4>后面，那么文章中的第一级标题从逻辑上就像<h5>一样，第二级标题从逻辑上就像<h6>一样，以此类推。

最后的结论是：HTML5有一个讲究逻辑的纲要机制，让组合文档变得更容易。在这种纲要机制下，标题的位置变得更加重要，而实际的级别则没有那么重要了。这样，可以免得你搬起石头砸到自己的脚。

2.5.4 解决一个纲要问题

到现在为止，你已经看到本章的示例，也看到了这些示例生成的纲要。而且，你看到的纲要也都非常符合规范。可是，有时候也会出现一个问题。比如，你创建了以下页面结构：

```
<body>
  <article>
    <h1>Natural Wonders to Visit Before You Die</h1>
    ...
    <h2>In North America</h2>
    ...
    <h3>The Grand Canyon</h3>
    ...
    <h3>Yellowstone National Park</h3>
    ...
    <h2>In the Rest of the World</h2>
    ...
    <aside>...</aside>
    ...
    <h3>Galapagos Islands</h3>
    ...
    <h3>The Swiss Alps</h3>
    ...
  </article>
</body>
```

然后，你觉得这个页面的纲要应该是这样的：

1. *Untitled Section for the* <body>
    1. Natural Wonders to Visit Before You Die
        1. In North America

        1. The Grand Canyon
        2. Yellowstone National Park
    2. In the Rest of the World
    3. *Untitled Section for the* `<aside>`
        1. Galapagos Islands
        2. The Swiss Alps

但实际的纲要却是这样的：

1. *Untitled Section for the* `<body>`
    1. Natural Wonders to Visit Before You Die
        1. In North America
            1. The Grand Canyon
            2. Yellowstone National Park
        2. In the Rest of the World
        3. *Untitled Section for the* `<aside>`
        4. **Galapagos Islands**
        5. **The Swiss Alps**

不知怎么地，<h2>后面多出来的那个<aside>把它后面的两个<h3>元素都带了出来，让它们在逻辑上跟<h2>元素排在了同一层次上。这显然不是你想要的结果。

为了解决这个问题，首先需要理解HTML5的纲要机制，即每次遇到编号标题元素（<h1>、<h2>、<h3>，等等），只要该元素不在某个区块顶部，就会为它自动创建一个新的区块。

对这个例子来说，纲要机制对一开头的<h1>元素什么都不做，因为它位于<article>区块的顶部。但纲要算法却会为接下来的<h2>和<h3>创建新的区块，就好像你写了如下标记一样：

```
<body>
  <article>
    <h1>Natural Wonders to Visit Before You Die</h1>
    ...
    <section>
      <h2>In North America</h2>
      ...
      <section>
        <h3>The Grand Canyon</h3>
        ...
      </section>
      <section>
        <h3>Yellowstone National Park</h3>
        ...
      </section>
    </section>

    <section>
      <h2>In the Rest of the World</h2>
      ...
```

```
      </section>
      <aside>...</aside>
      ...
      <section>
      <h3>Galapagos Islands</h3>
        ...
      </section>
      <section>
        <h3>The Swiss Alps</h3>
        ...
      </section>
    </article>
  </body>
```

多数情况下，这些自动创建的区块不是问题。事实上，这些区块通常还很有用，因为这样可以确保未正确编号的标题仍然能排在正确的纲要级别上。但这样做的代价就是偶尔会出现小差错，就像我们这个例子一样。

从生成的纲要来看，一开始都没有问题。顶部的<h1>自己一个级别（因为它已经在一个<article>里面了），接下来会为第一个<h2>元素创建一个子区块，然后再为这个子区块里面的<h3>分别创建各自的子区块，以此类推。当纲要算法碰到<aside>元素时，问题就出现了。既然碰到了<aside>，那就说明要关闭当前为<h2>创建的子区块了。而关闭这个子区块之后，再为<aside>后面的<h3>创建子区块，就会导致它们在逻辑上与前面的<h2>在纲要中处于同一个层次上。

为了纠正这个问题，需要通过自己定义区块和子区块，来代替纲要机制的自动创建行为。对我们这个例子来说，就是要避免第二个<h2>区块被过早地关闭。在标记中明确为该<h2>定义一个区块即可解决问题：

```
  <body>
    <article>
      <h1>Natural Wonders to Visit Before You Die</h1>
      ...
      <h2>In North America</h2>
      ...
      <h3>The Grand Canyon</h3>
      ...
      <h3>Yellowstone National Park</h3>
      ...
      <section>
        <h2>In the Rest of the World</h2>
        ...
        <aside>...</aside>
        ...
        <h3>Galapagos Islands</h3>
        ...
        <h3>The Swiss Alps</h3>
        ...
      </section>
    </article>
  </body>
```

这样，纲要算法就不必再为第二个<h2>自动创建区块了，因而也就避免了在它发现<aside>时去关闭该区块的风险。虽然也可以为文档中的所有标题都定义区块，但没有必要因此把标记搞乱，毕竟像这样修改一个地方就可以解决问题。

注意 另一个办法是用<div>替换<aside>。因为<div>不是分块元素，因此不会导致前面的区块意外关闭。

使用<aside>元素并不会经常导致这种问题。比如，前面文章中使用<aside>包含醒目引文，不就挺好嘛；那是因为<aside>正好在两个<h2>元素之间。可是，假如你不小心在两个不同级别的标题之间插入了一个分块元素，那就要检查一下纲要，看这样干是不是说得过去。

提示 如果你对本节讨论的所有纲要相关的概念还有点迷糊，不必担心。说老实话，纲要这件事是很多Web开发人员都不会十分关心的（至少目前是这样）。最好是把HTML5的纲要机制当成一种质量保障工具，它有时候可以帮上你的忙。通过在纲要生成器（参见2.5.1节给出的工具）中查看自己的标记，有可能发现其他问题导致的错误，而且能够确保你正确地使用语义元素。

第 3 章

编写更有意义的标记

上一章，我们了解了HTML5的语义元素。通过使用这些语义元素，可以为网页创建清晰、有逻辑的结构，而且能够适应未来更聪明的浏览器、搜索引擎和辅助设备。

不过，关于语义的故事并没有讲完。语义就是为标记赋予某种意义，然后可以在不同的标记中放入不同的信息。第2章谈到的语义元素完全是*页面结构*方面的，即用它们传达页面布局中大一些的内容块及所有区块的用途。但*文本级信息*同样也有语义，这样的语义元素用于描述较小一些的内容片段。可以使用文本级语义元素标注出重要的信息，以免它们被淹没在一望无际的网页大海之中。比如，可以为名字、地址、活动清单、产品、处方、餐馆评价加注语义元素。然后，各种各样的设备和工具——从新潮的浏览器插件，到专业的搜索引擎，便可各取所需。

本章，我们先来回顾一下列入HTML5语言规范的几个语义元素。之后，你还会学到一些文本级语义元素，在今天使用它们没有什么困难。接下来，我们会介绍几个致力于解决文本级语义问题的前沿标准。换句话说，我们会详细说一说*微数据*（microdata），它最初是作为HTML5规范的一部分被提出来的，但现在已经成了一个独立的标准，由W3C负责管理和推进。使用微数据，可以让网页和搜索结果的内容更加丰富。

3.1　回顾语义元素

我们从页面结构相关的元素（表3-1是一个简单的回顾）开始讨论语义是有原因的。原因也很简单，页面结构容易把握。毕竟，绝大多数网站在规划布局时，都会使用为数不多的几种常用设计元素（页眉、页脚、侧边栏、菜单），结果网站的布局——除了外观装饰有所不同外——都很相似。

表3-1　页面结构相关的语义元素

元　　素	说　　明
<article>	表示一篇任何形式的文章，即类似新闻报道、论坛帖子或博客文章（不包括评论或作者简介）等能够独立的内容区块
<aside>	表示独立于页面主内容的一个完整的内容块。例如，可以用<aside>创建一个附注栏，其中包含与主文章相关的内容或链接
<figure>和<figcaption>	表示一幅插图。其中<figcaption>元素标注图题（插图的标题），而<figure>元素标注<figcaption>和插入图片的<img>元素。目标是反映图片与图题之间是关联的

（续）

元　素	说　明
<footer>	表示页面底部的页脚[1]。通常是很小的一块内容，包括小字号的版权声明、简单的链接（比如About Us、Get Support等）
<header>	表示增强型的标题，可以包含HTML标题和其他内容。其他内容可以是标志、作者署名或一组指向后面内容的导航链接
<nav>	表示页面中重要的一组链接。其中的链接可以指向当前页面的主题，也可以指向网站的其他页面。实际上，一个页面中包含多个<nav>也很正常
<section>	表示文档中的一个区块，或者表示一组文档。<section>是一个通用容器，只有一条规则：其中的内容必须开始于一个标题。应该在其他语义元素（如<article>和<aside>）不适用的情况下再选用<section>
<main>	表示页面的主内容。比如，可以使用<main>包含<article>元素，隔离网站的页眉、页脚和侧边栏。这个元素是HTML5.1新增的

文本级语义可不是一块好啃的骨头。因为人们产生内容的类型千差万别。如果HTML5为每一种可能的内容类型都发明一个元素，那这门语言中包含的元素可就数不胜数了。问题的复杂性在于，结构化的内容是由很多更小的内容片段构成的，而组织这些内容片段的方式又没有一定之规。比如，要在页面中插入最简单的邮政地址，都可能需要好几个元素，比如<address>、<name>、<street>、<postalcode>、<country>，等等。

HTML5的策略是双管齐下。一方面，它只增强了很少几个文本级语义元素；另一方面，更重要的是，HTML5支持一个独立的微数据标准。这个标准为人们提供了扩展的可能性，任何人都可以定义自己的信息，在自己的页面中为信息添加标注。本章会介绍这两方面的主题。首先，我们来认识三个新的文本级的语义元素：<time>、<output>和<mark>。

3.1.1　使用<time>标注日期和时间

网页中经常会出现日期和时间信息。例如，绝大多数博客文章的末尾都有发表时间。然而，人们却一直没有标准的方式来标注日期，因此其他程序（如搜索引擎）也不容易提取这些信息，基本上都是靠猜。好了，现在<time>元素可以解决这个问题。通过它可以标注日期、时间或日期与时间的组合。先看一个例子：

```
The party starts <time>2014-03-21</time>.
```

注意　用<time>元素来标注（不包含时间的）日期似乎有点违反直觉，我告诉你吧，这只是HTML5的古怪行为之一。更恰当的元素显然应该是<datetime>，但他们硬是不用。

<time>元素在这里要扮演两个角色。首先，它表示日期和时间位于标记中的哪个地方。其次，它以任何软件程序都能理解的方式提供日期和时间。前面的例子符合第二个角色的要求，使用了

注1：根据HTML5规范，<footer>元素不仅可以用于页面的页脚，也可以用于文章中的文脚。（译者注）

通用的日期格式，即按年月日这个顺序，年4位数、月和日均为2位数，分隔符为短划线。换句话说，日期的格式如下：

```
YYYY-MM-DD
```

不过，对于浏览网页的人，你可以随便采用任何格式来显示日期。实际上，任何文本形式也都是允许的，只要你在datetime属性中提供计算机能看懂的通用格式的日期就行，比如：

```
The party starts <time datetime="2014-03-21">March 21<sup>st</sup></time>.
```

在浏览器中的效果如下所示：

The party starts March 21st.

对于时间部分，<time>也有类似的规则，标准的时间格式如下：

```
HH:MM
```

首先是2位数的小时（24小时制），然后是2位数的分钟。比如：

```
Parties start every night at <time datetime="16:30">4:30 p.m.</time>.
```

最后，组合以上两个规则可以指定具体日期和时间，日期在前，空一个格，时间在后：

```
The party starts <time datetime="2014-03-21 16:30">March 21<sup>st</sup>
at 4:30 p.m.</time>.
```

注意 最初，<time>元素要求用一种稍有不同的格式组合日期和时间。区别在于不用空格分隔符，而用大写的T（表示time），比如2014-03-21T16:30。这种格式还可以用，因此在别人的网页也可能出现。

在组合日期和时间的情况下，有时候还需要后缀时区信息。比如，纽约在东五区，即UTC-5:00。（关于时区划分，请参见http://en.wikipedia.org/wiki/Time_zone。）要表示纽约时间下午4:30，应该使用如下标记：

```
The party starts <time datetime="2014-03-21 16:30-05:00">March 21<sup>st</sup>
at 4:30 p.m.</time>.
```

这样，看你的网页的人，能看到他们想看到的格式，而搜索机器人和其他软件则能看到它们可以处理的值，而且毫无歧义。

另外，<time>还有一个pubdate属性。如果当前内容（例如<time>元素所在的<article>）对应一个发表日期，可以使用这个属性。下面就是一个例子：

```
Published on <time datetime="2014-03-21" pubdate>March 31, 2014</time>.
```

注意 因为<time>元素纯粹是信息性的，没有任何附加的样式，所以可以在任何浏览器中使用。不必担心兼容性问题。不过，假如你想给<time>元素添加点样式，需要针对Internet Explorer使用2.3节介绍的方法。

3.1.2 使用<output>标注JavaScript返回值

HTML5还包含一个语义元素<output>，它能使某种JavaScript驱动的页面更加清晰。实际上，这个元素就是一个占位符，用于展示一小段计算后的信息。

例如，假设你要创建如图3-1所示的一个页面。用户可以在这个页面中输入某些信息。然后用脚本取得这些信息，进行计算，并将结果显示在下面。

通常的做法是给占位符指定一个ID属性，这样JavaScript代码就可以在计算时找到它。Web开发人员一般将<span>元素用做占位符，而唯一的问题就是该元素不提供任何语义：

3

```
<p>Your BMI: <span id="result"></span></p>
```

以下则是使用HTML5的更有意义的版本：

```
<p>Your BMI: <output id="result"></output></p>
```

实际的JavaScript代码无需任何改变，因为它只根据ID属性查找元素，不考虑元素类型：

```
var resultElement = document.getElementById("result");
```

注意 在使用<output>之前，别忘了像2.3节介绍的那样为Internet Explorer包含相应的脚本。否则，在旧版本的IE（IE8及更早版本）中无法通过JavaScript来访问该元素。

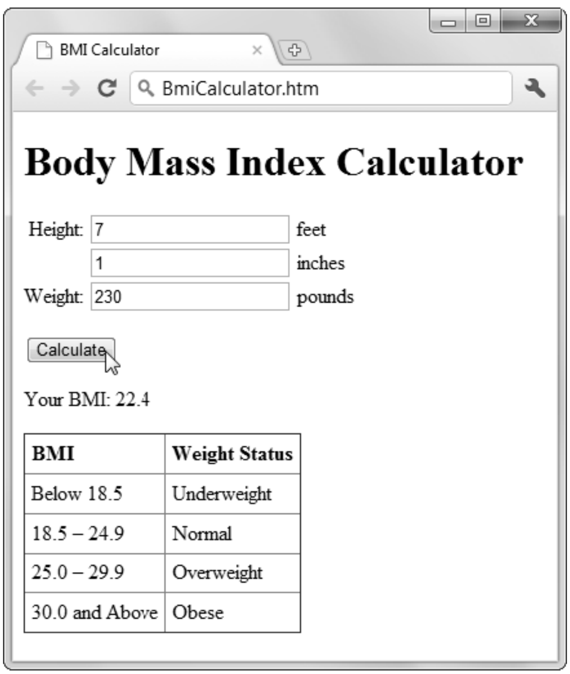


图3-1：这是一个由来已久的设计模式，输入数字，单击按钮，就可以得到答案

通常，这种页面都会采用一个<form>元素。在这个例子中，需要包含3个文本框，以便用户在其中输入信息：

```
<form action="#" id="bmiCalculator">
  <label for="feet inches">Height:</label>
  <input name="feet"> feet<br>
  <input name="inches"> inches<br>

  <label for="pounds">Weight:</label>
  <input name="pounds"> pounds<br><br>
  ...
</form>
```

如果你想让自己的<output>元素变得更聪明一些，可以为它添加form和for属性，前者的值是包含相关控件的表单ID，后者的值是以空格分隔的相关控件的ID。比如，下面就是一个例子：

```
<p>Your BMI: <output id="result" form="bmiCalculator"for="feet inches pounds">
</output></p>
```

这些属性实际上什么也不干，唯一的用处就是表明<output>从哪些控件获取结果这一信息。不过，你肯定会因为考虑到了语义而获得加分。如果其他人需要编辑你的页面，那这些属性可以帮他们理清思路。

提示 如果你对表单还不很熟悉，第4章会详细介绍。假如你对JavaScript还不如你对世界语了解得多，那可以先通过附录B复习一下。而假如你想自己试一试这个页面，可以在http://prosetech.com/html5中找到所有示例页面。

3.1.3 使用<mark>标注突显文本

<mark>元素用于标注一段文本，这段文本会突出显示。在需要引用其他人的内容，而你想引起别人注意时，就可以使用<mark>元素：

```
<p>In 2009, Facebook made a bold grab to own everyone's content,
<em>forever</em>. This is the text they put in their terms of service:</p>
<blockquote>You hereby grant Facebook an <mark>irrevocable, perpetual,
non-exclusive, transferable, fully paid, worldwide license</mark> (with the
right to sublicense) to <mark>use, copy, publish</mark>, stream, store,
retain, publicly perform or display, transmit, scan, reformat, modify, edit,
frame, translate, excerpt, adapt, create derivative works and distribute
(through multiple tiers), <mark>any user content you post</mark>
...
</blockquote>
```

如图3-2所示，浏览器会给<mark>中的文本添加黄色背景。

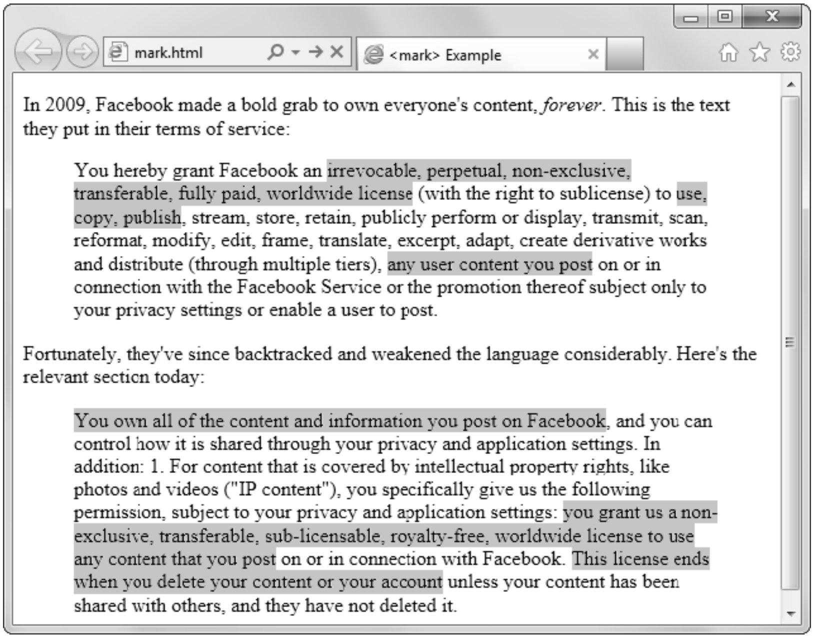


图3-2：在这个例子中，我们使用<mark>元素突出显示了引用的一段话中的重要细节

也可以使用<mark>标注重要的内容或关键字——就像搜索引擎在搜索结果中显示匹配的文本那样，还可以与<del>（删除的文本）和<ins>（插入的文本）组合使用，以标注文档的变化。

实话实说，<mark>元素其实并不十分必要。HTML5规范认为它是一个语义元素，但实际上它更多地则是被用于表现目的。默认情况下，被标注的文本会带有浅黄色背景（见图3-2），不过当然可以通过样式表规则为它应用不同的样式。

提示 <mark>元素的目的并不在于样式。毕竟，突出显示文本的方式已经有很多了。因此，应该坚持只用<mark>（结合你想使用的任意CSS格式）来传达适当的语义。这里有一条经验，那就是使用<mark>来吸引人注意那些变得很重要的文本。比如，关于写作该文本的讨论，或者它是用户应该执行的任务。

即使你想沿用默认的黄色背景，也应该为不支持HTML5的浏览器添加后备样式表。以下就是相应的样式规则：

```
mark {
  background-color: yellow;
  color: black;
}
```

为了让<mark>在旧版本的IE中能够正常获得样式，还应该像2.3节介绍的包含相应的脚本文件。

3.2 其他语义标准

此时，你可能会想，HTML中还缺少很多语义元素。没错，现在可以标注日期、突出显示文本，但对其他常见的信息怎么办，比如，名字、地址、企业名录、产品说明、个人简介？HTML5有意不在这方面投入精力，因为其制定者不想把这门语言过分地特殊化，否则就避免不了“有人欢喜有人忧”的怪圈。如果真想在语义方面有更大的作为，就必须跳出HTML5语言核心的范畴，去寻找几种更适合你网页的标准。

通过标记巧妙地表达语义并不是什么新点子。事实上，早在HTML5还是一个幻想，存在于WHATWG编辑Ian Hickson的大脑中的时候，就有非常多的Web开发人员大声疾呼，要求为他们提供更有意义的标记。他们的目标也不尽相同，有些人希望提高无障碍性，有些人打算要进行数据挖掘，而还有一些人仅仅是希望自己的简历多一个闪光点。不过，所有这些人在标准HTML语言中都找不到自己想要的标记。由此，一些新标准的应运而生也就不足为奇了。

接下来几节，我们介绍4种这样的标准。首先，就是ARIA，它致力于提升屏幕阅读器中的无障碍性。其次，我们要看一下另外三种并存的手段，它们用于描述不同的内容，比如联系人、地址、企业名录或其他各种HTML页面可能出现的内容。

3.2.1 ARIA

ARIA（Accessible Rich Internet Application，无障碍富因特网应用）是一个还在制定中的标准，它规定了在任意HTML元素上使用的属性，而通过这些属性可以为屏幕阅读器提供额外的信息。例如，ARIA中规定了一个`role`属性，表示所在元素的用途。下面就以一个表示页眉的`<div>`元素为例：

```
<div class="header">
```

通过为其指定值为`banner`的`role`属性，可以告诉屏幕阅读器它的用途是保存横幅广告：

```
<div class="header" role="banner">
```

当然，我们上一章刚刚学过，HTML5为标注页眉提供了一个语义元素。因此，这时候最合适的方式莫过于：

```
<header role="banner">
```

这个例子说明了两方面问题：首先，ARIA规定了一些推荐的角色名。（要了解所有角色名，请参考规范中相应的部分：http://tinyurl.com/roles-aria。）其次，部分ARIA与新HTML5语义元素重复——这是合情合理的，因为ARIA早于HTML5。不过，重复也不是完全重复。比如，有些角色名在HTML5中确实也出现了（像“banner”和“article”），而有些则可能在将来才会出现（如“toolbar”和“search”）。

ARIA还针对HTML表单定义了属性。文本框的`aria-required`属性表示用户必须输入值。而文本框的`aria-invalid`属性表示当前值不正确。这些属性可以让屏幕阅读器用户，特别是视力不好的用户，不会因为看不到提示信息（比如必填字段旁的星号，或闪烁的红色错误图标）而受影响。

为了恰当地采用ARIA，需要学习标准，也要花时间评审自己的标记。对于是否值得为此投入，Web开发人员的看法不一。因为这个标准尚未制定完成，而HTML5又提供了一些类似的语义元素，且用起来省事儿不少。不过，假如你真的想创建一个可以无障碍访问的网站，必须既要考虑ARIA，又要考虑HTML5。因为屏幕阅读器支持ARIA，不支持HTML5。

注意 要了解有关ARIA（全名是WAI-ARIA，因为它是由Web Accessibility Initiative工作组负责制定的）的更多信息，请参考其规范：http://www.w3.org/TR/wai-aria/。

3

3.2.2 RDFa

RDFa（Resource Description Framework，资源描述框架）也是一种使用属性向网页中嵌入详细信息的标准。与本章讨论的其他方式不同，RDFa有一个明显的优点：它是一个稳定不变的标准。RDFa也有两个明显的缺点。首先，RDFa最初是为XHTML而非HTML5设计的，因此关于严格的RDFa与松散的HTML5如何混写的争论一直不绝于耳。其次，RDFa很复杂，嵌入RDFa元数据后的网页比最初要长得多，而且显得很笨重。由于这种复杂性，RDFa"藏污纳垢"的可能性就增大了。根据谷歌的调查，其包含错误的概率大约高三倍。

本章不会详细讨论RDFa（除了下一节介绍的与HTML5密切相关的微格式），不过假如你希望了解更多信息，可以在Wikipedia上看到完整的介绍：http://en.wikipedia.org/wiki/RDFa。或者，也可以参考3.4.1节介绍的Google Rich Snippets，其中所有示例都有对应的RDFa版本。

3.2.3 微格式

微格式（Microformats）是一种在网页中嵌入元数据的简单而又比较合理的方式。微格式并没有想成为任何官方标准。相反，它是一组统一的宽松的约定，这些约定有助于网页之间共享结构化信息，但又不会导致像采用RDFa那样复杂。微格式因此获得了巨大的成功，最近的一次调查显示，在包含丰富元数据的页面中，有70%都基于微格式。

微格式的应用方式比较新颖，它们附加在通常用于添加样式的`class`属性上。你可以根据数据的类型，使用某些标准的样式名来标注数据。然后，其他程序可以读取你的标记，提取数据并通过检查`class`属性来确定数据的含义。

举个例子，可以使用hCard表示一个人、一家公司或组织，或者一个地方的详细联系信息。第一步是添加一个包含正确类名的根元素。对hCard而言，类名是`vcard`。（一般来说，类名与格式名是匹配的，这里之所以叫`vcard`有个历史原因。过去有一种格式叫Versitcard，hCard以这种格式为基础，因此就有了`vcard`这个类名。）

下面这个`<div>`元素包含了hCard，可以用于包含详细联系信息：

```
<div class="vcard">
</div>
```

在这个根元素中，可以添加联系人信息。每个信息都要单独用一个元素包含，同时赋予该元素一个正确的类名。比如，在hCard中，可以使用fn这个类名表示人的全名，使用url表示个人主页：

```
<div class="vcard">
  <h3 class="fn">Mike Rowe</h3>
  You can see Mike Rowe's website at
  <a class="url" href="http://www.magicsemantics.com">www.magicsemantics.com
  </a>
</div>
```

在微格式中使用类名时，不需要在样式表中创建对应的样式。拿上面的例子来说，就是不必给vcard、fn或url写样式规则。此时类名的用处，就是告诉别人你的数据结构良好，意思明确。

注意 标记数据前，要先选好想用的微格式。可选的不多，大多数还在修订和改进。至于有哪些可选，以及它们的详细用法，请参考：http://microformats.org/wiki。要了解更多关于hCard的信息，请参考http://microformats.org/wiki/hcard。

只要会使用hCard，再使用hCalendar就毫无压力了。后者的流行程度仅次于hCard。可以使用hCalendar标注约会、会议、假日、产品发布时间、商店开业时间等。首先，需要把活动列表包装在一个元素中，将这个元素的class属性值设置为vevent。在这个元素内部，至少要包含两段信息：开始日期（使用类名dt-start）和说明（使用类名summary）。此外，也可以选择http://microformats.org/wiki/hcalendar中列出的其他可选的属性，包括结束日期工持续时间、地点，以及包含更详细信息的页面URL。下面来看一个例子：

```
<div class="vevent">
  <h2 class="summary">Web Developer Clam Bake</h2>
  <p>I'm hosting a party!</p>
  <p>It's
  <span class="dtstart" title="2014-10-25 13:30">Tuesday, October 25,
  1:30PM</span>
  at the <span class="location">Deep Sea Hotel, San Francisco, CA</span></p>
</div>
```

看到微格式这么流行，你恐怕会认为语义Web争夺战应该可以尘埃落定了。可惜还没有，听我说。首先，数量极为庞大的网页中包含的标记仍然毫无语义可言。其次，就算采用了微格式的网页，其用途也只有两个：联系信息和活动列表。既然微格式的应用还十分有限，那这个领域的竞争就不可避免。再次，潮流已经开始转向不太为人所知但却更加灵活的微数据规范。人们发现，微格式似乎只是走向微数据标准的一个中间停靠站。下一节我们就介绍微数据。

3.2.4 微数据

微数据（Microdata）是尝试解决语义标记问题的另一种选择。它最早是HTML5规范的一部分，后来分离出来成为一个独立的标准：http://dev.w3.org/html5/md/。微数据采用的方法与RDFa

类似，但简化了一些。另外，与微格式不同，它使用自己的属性，因而不存在与样式表规则冲突的风险（也不会让Web开发人员感到困惑）。这种设计意味着微数据更讲究逻辑性，而且更容易改编为适合你网页的语言。不过这一切是以牺牲简洁性为代价的，采用微数据的网页要比采用微格式的网页大很多。

最近，随着微软、谷歌、雅虎和Yandex（俄罗斯最大搜索引擎）组团创建微数据收录网站http://schema.org，微数据获得了空前的发展。在前面提到的网站中，可以看到各种微数据格式和示例，包括 Person和Event（效仿的是hCard和hEvent微格式），以及更特定的格式，比如针对公司、餐馆、评审、产品、图书、电影、菜谱、电视节目、公交站点、景点、健康、药物等的格式。目前，只有搜索引擎关注这种信息，但它们驱动流量和塑造Web的威力是任何人都不能小看的。（3.4节将介绍搜索引擎如何使用这种信息。）

注意 现在看来，微数据作为元数据的一种适当标准，有望走红。这是因为它比微格式灵活，比RDFa简单。

在标注微数据时，首先要给相应元素（如果连元素也需要新添加，那<div>元素从逻辑上更像一个容器）添加itemscope和itemtype属性。其中，itemscope属性表示开始一段新的语义内容，而itemtype属性表示内容中包含的数据类型：

```
<div itemscope itemtype="http://schema.org/Person">
</div>
```

标识数据类型时，要使用一个叫XML命名空间的预定义的、独一无二的字符串。在这个例子中，http://schema.org/Person就是XML命名空间，其中定义了用于编写联系人信息的标准的微数据格式。

微数据的命名空间

每一种微数据都需要一个命名空间。从技术角度说，命名空间用于标识微数据所用的词汇。比如，命名空间http://schema.org/Person就表示当前标记要使用Person词汇表。在http://schema.org中，可以看到几十种微数据的词汇表（见图3-3）。

XML命名空间一般是URL形式的。有时候，在浏览器中打开相应的URL，可以看到对相应数据类型的说明（就像你打开http://schema.org/Person一样）。不过，XML命名空间不一定非要在网上有对应的页面，甚至不必是URL形式。换句话说，是什么形式，完全取决于创建相应格式的人。URL形式的好处是利用个人或公司的域名，从而确保命名空间的唯一性。这样，就不应该有其他人使用同一个命名空间来创建不同的数据格式，也就避免了混淆。

如果命名空间以http://schema.org开头，说明它得到了微软、谷歌、雅虎和Yandex的支持。因此，如果你使用它的词汇表，那可以放心，世界上最大的搜索引擎都能理解你在干什么。如果命名空间以http://data-vocabulary.org开头，那说明其词汇表稍老一些。虽然大多数搜索引擎也能理解，但最好还是与时俱进，到http://schema.org中去找等价的词汇表。

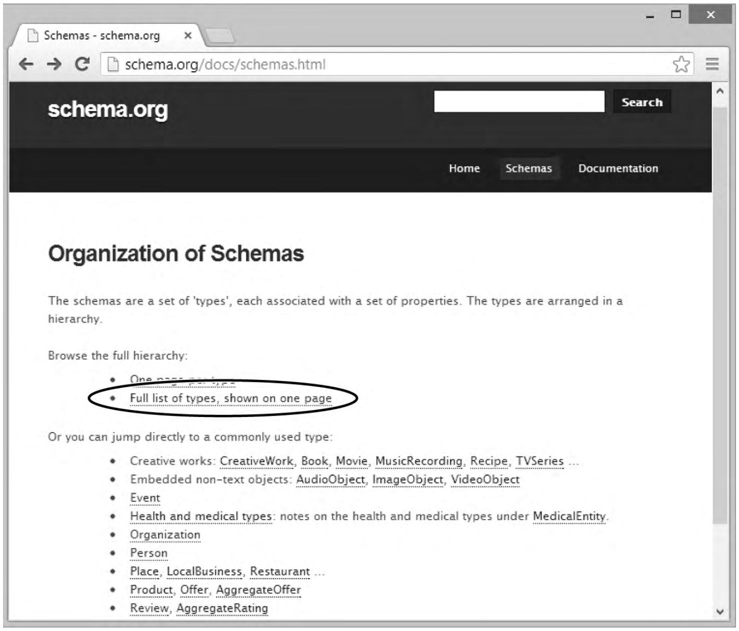


图3-3：要找到合用的微数据词汇表，最好从http://schema.org/docs/schemas.html找起。单击链接可以跳到常用词汇表的定义，比如Person、Organization、Event。或者单击“Full list of types”链接查看各种分类

定义完容器元素后，就可以开始下一步了。在容器元素内，使用itemprop属性来标注重要信息。这种方式与微格式的基本规则相同——通过公认的itemprop名称，其他软件可以从关联的元素中取得信息。

以下代码使用了微数据，相当于前面使用微格式的标记：

```
<div itemscope itemtype="http://schema.org/Person">
  <h3 itemprop="name">Mike Rowe</h3>
  You can see Mike Rowe’s website at
  <a itemprop="url" href="http://www.magicsemantics.com">www.magicsemantics.
  com</a>
</div>
```

注意 微数据与微格式最重要的不同就是，微数据标注元素时使用itemprop属性，而不是class属性。由于微数据使用自己的itemcope、itemtype和itemprop属性，而不是class属性，因此你的语义标记不会与样式表格式混淆。

使用Person词汇表可以标注很多细节，比如邮政地址、电子邮箱、电话号码、生日、照片、职务、单位名称、性别、国籍，等等。全面的信息请参考：http://schema.org/Person。

注意 这三种富语义数据标准——RDFa、微数据和微格式——很多地方都是相似的。虽然它们不完全兼容，但使用其中一种标准的技能，多数情况下都适用于其他标准。

3.3　实例：改进“About Me”页面

至此，我们已经了解了微格式和微数据这两个语义流派的基本结构。以此为起点，大家可以自己（到http://microformats.org）去找新的微格式，或者（到http://schema.org）去找微数据词汇表，从头开始编写富有语义的标记。

但生活往往并没有那么如意，至少对大多数Web开发者来说没有。更常见的情况则是你手头已经有了一个包含数据的网页，你需要改进其中的标记，将其语义化。如果你记住以下几点，这件事倒也不难。

3

- 通常，有些重要的数据会混在一些无关紧要的数据中。此时，可以给重要的数据加上新标记，比如用<div>标记块级元素，用<span>标记行内元素。
- 不用担心信息的次序。只要使用的类名（微格式）或属性名（微数据）正确，就可以按照自己的意愿组织标记。
- 想显示图片，就用<img>；想放个链接，就用<a>。除此之外，一般就是标记普通的文本。

下面就有一个典型的例子。图3-4中的“About Me”页面包含以下内容：

```
<h1>About Me</h1>

<img src="face.jpg" alt="Mike's Face">
<p>This website is the work of <b>Mike Rowe Formatte</b>.
His friends know him as <b>The Big M</b>.</p>

<p>You can contact him where he works, at
The Magic Semantic Company (phone
641-545-0234 and ask for Mike) or email mike-f@magicsemantics.com.</p>

<p>Or, visit Mike on the job at:<br>
42 Jordan Gordon Street, 6th Floor<br>
San Francisco, CA 94105<br>
USA<br>
<a href="http://www.magicsemantics.com">www.magicsemantics.com</a>
</p>
```

很明显，这个例子用我们熟悉的Person词汇表（http://schema.org/Person）正合适。我们试着给“About Me”页面中的重要信息添加微数据标记。新增的微数据标记都加粗了。

```
<h1>About Me</h1>

<div itemscope itemtype="http://schema.org/Person">
  <img itemprop="image" src="face.jpg" alt="Mike's Face">
  <p>This website is the work of
  <span itemprop="jobTitle" style="display:none">Web Developer</span>
  <b itemprop="name">Mike Rowe Formatte</b>.
  His friends know him as <b itemprop="additionalname">The Big M</b>.</p>
  <p>You can contact him where he works, at
  The Magic Semantic Company</span> (phone
  <span itemprop="telephone">641-545-0234</span> and ask for Mike)
  or email <span itemprop="email">mike-f@magicsemantics.com</span>.</p>
```

```
  <p>Or, visit Mike on the job at:<br>
  42 Jordan Gordon Street, 6th Floor<br>
  San Francisco, CA 94105<br>
  USA<br>
  <a itemprop="url" href="http://www.magicsemantics.com">www.magicsemantics.
com</a>
  </p>
</div>
```

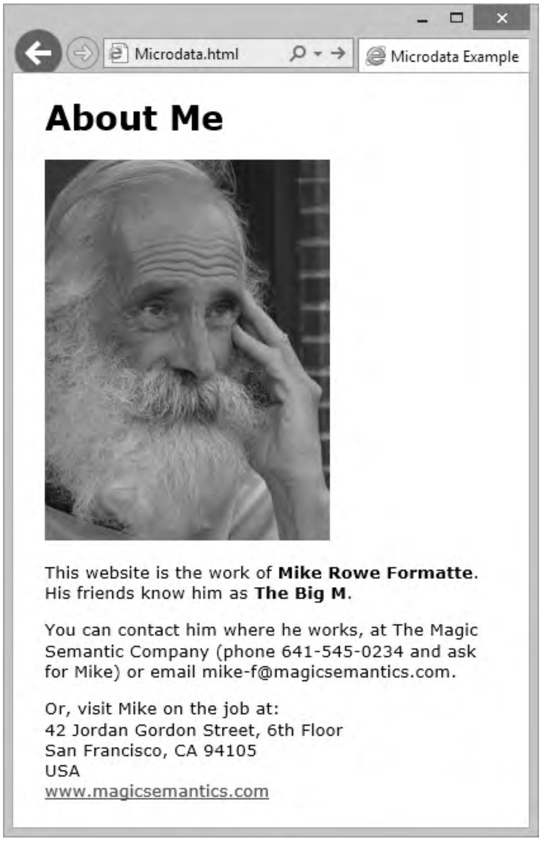


图3-4：这个“About Me”页面包含网站作者的详细联系信息，但尚未添加任何微数据标记

这个例子用了几种技巧。

- 使用新包含了需要标注微数据的内容。
- 给原有合适的元素添加了itemprop属性。比如，里面是名字，因此没必要再多此一举弄个了。（当然，你加上也可以。就像这样：Mike Rowe Formatte。）
- 使用看不见的<span>表示人的职务。（文本使用行内样式将display设为none。）这个技巧经常用于隐藏多余信息，但又可以让搜索引擎和其他工具找得到。也有人说这种做法会导致一些工具（如Google）对其视而不见，因为这些信息对用户不可见。

微数据经常会有嵌套的结构，即一个词汇表会套住另一个。比如，可以在Person词汇表的个人详细信息中套上地址信息，而地址信息使用的词汇可能来自PostalAddress词汇表。

标记地址信息时，要使用带有itemprop、itemscope或itemtype属性的<div>或<span>元素。itemprop是属性名，itemscope属性表示要使用新词汇表提供属性数据，而itemtype属性是以XML命名空间表示的词汇表（在这里是http://schema.org/PostalAddress）。下面是两个词汇表嵌套的结果：

```
<div itemscope itemtype="http://schema.org/Person">
  <img itemprop="image" src="face.jpg" alt="Mike's Face">
  <p>This website is the work of
  ...
  <p>Or, visit Mike on the job at:<br>
  <span itemprop="address" itemscope
   itemtype="http://schema.org/PostalAddress">
    ...
  </span>
</div>
```

接下来就可以在新区域中标注地址信息了：

```
<div itemscope itemtype="http://schema.org/Person">
  <img itemprop="image" src="face.jpg" alt="Mike's Face">
  <p>This website is the work of
  ...

  <p>Or, visit Mike on the job at:<br>
  <span itemprop="address" itemscope
   itemtype="http://schema.org/PostalAddress">
    <span itemprop="streetAddress">42 Jordan Gordon Street,
     6th Floor</span><br>
    <span itemprop="addressLocality">San Francisco</span>,
    <span itemprop="addressRegion">CA</span>
    <span itemprop="postalCode">94105</span><br>
    <span itemprop="addressCountry">USA</span><br>
  </span>
  ...
</div>
```

这样完全没有问题。或许你不太清楚什么时候可以在一个微数据区域中再新开一个微数据区域，那就看看这个参考页面吧：http://schema.org（参见图3-5）。

在标注公司名称时，同样可以采用这种嵌套的方式 。在此，需要把人的affiliation属性设置为使用Organizatioin词汇表：

```
<p>You can contact him where he works, at
<span itemprop="affiliation" itemscope
 itemtype="http://schema.org/Organization">
  <span itemprop="name">The Magic Semantic Company</span>
</span>
```

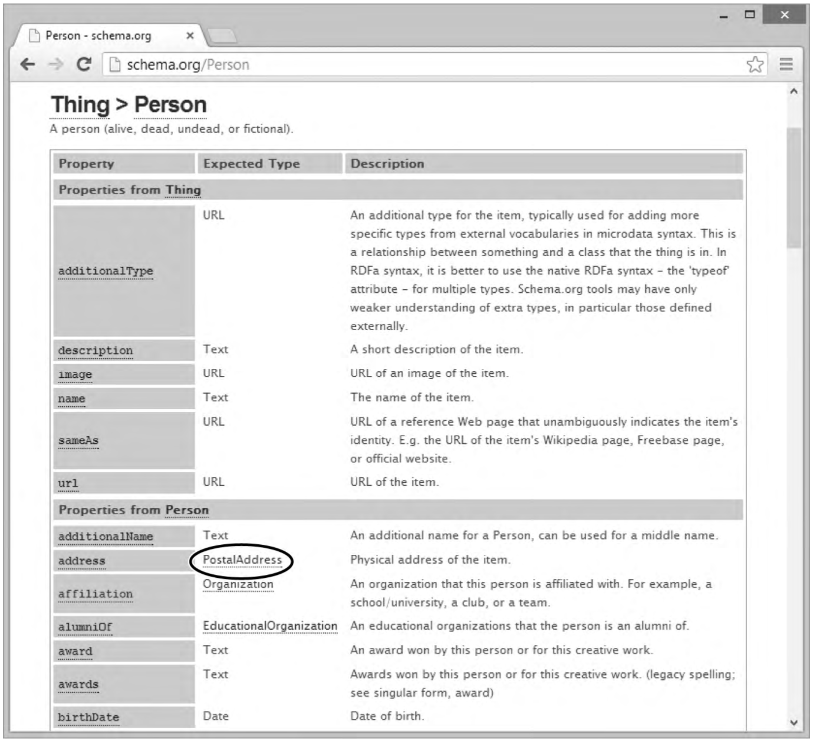


图3-5：Person词汇表的属性很多（图中只显示了一少部分）。大多数属性用于标注普通文本、数值或日期。但有一些（如这里能看到的address属性）会使用自己的词汇表（PostalA ddress）。单击相应链接，可以了解更多内容

提示　如果你不喜欢手工输入itemtype或itemprop等属性名，可以找一个在线工具来帮你自动生成。比如，http://schema-creator.org和http://www.microdatagenerator.com。这两个网站的用法差不多，你选择词汇表，然后在文本框里输入数据，最后复制生成的标记。

在浏览器中提取语义数据

说到这儿，我们还应该了解一下这么做的好处。尽管目前还没有浏览器能原生识别微格式（至少在本书写作时没有），但还是有不少插件和脚本可以赋予浏览器这种能力。至于这些语义数据的用途，当然也不少。比如，浏览器可以检测页面中的联系信息，把它们单列在一个面板中，让你像添加页面书签一样，迅速地添加联系人信息。再比如，浏览器检测到会议信息后，你可以只单击一下，就把该会议添加到自己的活动日历中，或者在地图中标出会议地点。

目前还没有插件能做到这些。不过，一些骨灰级的Web开发人员已经写了不少JavaScript脚本，用于搜索微格式或微数据，通过弹出层显示它们，或者用于其他用途。（比如用JavaScript写的微格式工具：http://krofdrakula.github.io/microdata-tool/。）某些浏览器也有相应的插件可以发现网页中不同的元数据（图3-6）。



图 3-6 ： Chrome 扩展 Semantic Inspector可以提取当前页面中的所有元数据。单击其工具条中的按钮，就可以看到语义数据层。这个图是使用该扩展提取“About Me”页面中元数据的结果

将来，主流浏览器很可能原生支持微格式，就像IE和Firefox原生支持数据源一样。数据源是一种特殊的标记文档，可以提供最新内容，比如最新文章汇总等。如果你使用Firefox打开某个博客，它会自动检测博客的RSS数据源，为你提供一个“实时”的新内容书签。微格式将来也可以为网页提供类似的附加价值。

3.4　搜索引擎如何使用元数据

在自己的页面里加入语义上的细节，是为你赢得Web极客称号的不错方式。不过，即使是最忠实的Web开发者，也要判断做这些额外的工作（以及把标记搞得不那么整洁）是否值得。如果所有浏览器都绝顶聪明，都能识别这些语义该有多好啊！但冷酷的现实是，只有屈指可数的几个不那么知名的浏览器插件可供使用。

好在，为标记添加丰富的语义还有一个理由：SEO（Search Engine Optimization，搜索引擎优化）。SEO是一种艺术，可以让你的网站更容易被搜索引擎曝光；换句话说，你的网页能够更频繁地出现在搜索结果中，对某些关键词能有更好的排名，更有可能吸引用户点击访问你的网站。完备的元数据可以提升用户点击访问你的网站的可能性。你要做的就是在页面中放入正确的语义数据，然后谷歌等搜索引擎可以找到它们并构建一个漂亮的搜索列表，让你的网站能鹤立鸡群。

3.4.1　Google Rich Snippets

今天，大多数搜索引擎都能理解自己收录的页面中的元数据。本章后面将主要讲解谷歌如何处理元数据。之所以讲谷歌，主要因为它是世界第一搜索引擎，占据三分之二的市场份额，另外也因为谷歌多年来一直使用和推动元数据。因此，谷歌今天使用元数据的方式，可能就是其他搜索引擎明天使用元数据的方式。

Rich Snippets是谷歌推出的一个工具，能够将RDFa、微格式和微数据结合起来。前面已经介

绍过了，这几种语义标准有很多共性，而且都是为了解决相同的问题。谷歌了解所有这些标准，并且尝试对它们一视同仁，因此无论你使用哪种标准，都没有问题。（接下来的例子将使用微数据，同时也是为了让你了解HTML5最新的语义标准。）

要了解谷歌支持的元数据的更多信息，请参考谷歌的文档：http://tinyurl.com/GoogleRichSnippets。这个页面不仅介绍了RDFa、微格式和微数据，还展示了很多不同的片段的示例（如联系人信息、活动、产品、评论、配方，等等。）更赞的是，谷歌在每个例子中都提供了RDFa、微格式和微数据的版本，通过这些例子，你可以全面掌握所有语义标准的使用方法。

3.4.2 增强搜索结果

要了解Google Rich Snippets如何工作，可以使用谷歌的Structured Data Testing Tool。这个工具会检测你提供的页面，展示谷歌从中提取出来的语义数据以及谷歌如何利用这些数据定制该页面在用户搜索结果中显示的方式。

注意 Structured Data Testing Tool的用处体现在两方面。首先，它能帮你验证语义标记。如果谷歌无法全部提取出你放到页面中的信息，或者其中某些信息被标记为错误的属性，就说明你可能有地方做错了。其次，它能向你展示语义数据对出现在谷歌搜索结果中页面外观的影响。

要使用Structured Data Testing Tool，可以参考如下步骤。

(1) 打开http://www.google.com/webmasters/tools/richsnippets。

这个页面中包含一个文本框（见图3-7）。

(2) 如果你想粘贴标记，单击HTML标签。使用Structured Data Testing Tool的方式有两种，由页面中的两个标签表示。

- URL标签。让谷歌分析在线页面，在这里粘贴完整的网址；
- HTML标签。可以粘贴你想分析的标记段（不需要完整的页面标记），适合还没有上线的页面。

(3) 输入URL或粘贴标记段，单击Preview按钮。

这样，就可以预览结果（见图3-7）。预览时要注意两部分，一是Google search preview部分，其中包含页面在谷歌搜索结果中的样子。另一个是Extracted rich snippet data from the page，其中包含谷歌能够从页面中提取出来的所有原始语义数据。

提示 如果你看到了“Insufficient data to generate the preview”（没有足够的数据生成预览），有三种可能。第一，你的标记可能有错误。此时，查看一下谷歌提取出来的原始数据，确保它找到了你放在页面中的所有数据。如果不是这个问题，那有可能是你使用了某个数据类型，而谷歌尚未支持该类型，或者你添加的属性还未达到谷歌要求的最低数目。实在不行，可以拿你的标记与谷歌的例子（http://tinyurl.com/GoogleRichSnippets）进行对比。



图3-7：在此，谷歌找到了联系人信息和地址信息（基于3.2.4节微数据的例子）；有了这些信息，谷歌在页面标题下方添加了一行灰色的文本，把联系人信息显示了出来

谷歌使用的强调联系人信息（图3-7）的方法比较受限制。不过，谷歌能识别的富数据类型仅限于联系人信息。本章前面，我们介绍了如何使用微格式定义活动（见3.2.3节）。如果你在页面中添加一系列活动，那谷歌可能会把相关信息显示在搜索结果下面，如图3-8所示。

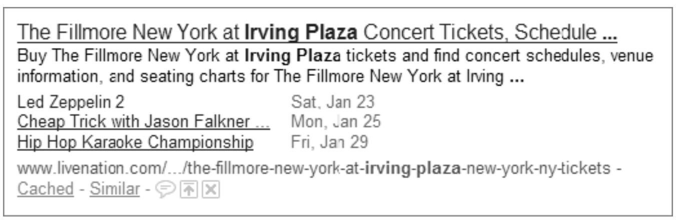


图3-8：这个示例页面中包含三个活动。如果你提供的URL中包含几个活动（像这里这样），那谷歌会把它们转换成可点击的链接

谷歌也能识别企业名录（处理方式与处理个人联系信息类似）、食谱（下一节介绍相关的例子）和评论（下面马上会讨论到）。

接下来的例子展示了一些标记，我们需要把其中的评论文本转换为可识别的评论微数据。相关的数据标准定义请参考http://schema.org/Review。其中关键的属性有`itemReviewed`（在这个例子中是餐馆）、`author`（留言的人）和`reviewBody`（评论全文）。此外，还可以给出一句话的概述（`description`）、撰写或提交评论的时间（`datePublished`，支持HTML5的`<time>`元素），以及从0到5的一个星级评分（`reviwRating`）。

以下就是一个例子，其中加粗了所有微数据：

```
<div itemscope itemtype="http://schema.org/Review">
  <p itemprop="description">Pretty bad, and then the Health Department showed
    up.
  </p>
<div itemprop="itemReviewed" itemscope itemtype="http://schema.org/Thing">
  <span itemprop="name">Jan's Pizza House</span>
</div>

<div>
  <span itemprop="author" itemscope itemtype="http://schema.org/Person">
    Reviewed by: <span itemprop="name">Jared Elberadi</span>
  </span>
  on <time itemprop="datePublished" datetime="2014-06-26">January 26</time>
</div>

<div itemprop="reviewBody">I had an urge to mack on some pizza, and this
 place was the only joint around. It looked like a bit of a dive, but I went
 in hoping to find an undiscovered gem. Instead, I watched a Health
 Department inspector closing the place down. Verdict? I didn't get to
 finish my pizza, and the inspector recommends a Hep-C shot.</div>

  <div itemprop="reviewRating" itemscope itemtype="http://schema.org/Rating">
   Rating:<span itemprop="ratingValue">1.5</span></div>
</div>
```

把这些用微数据格式化的信息放到页面中，谷歌会给出特殊处理（图3-9）。

眼力好的读者会发现，这一篇评论中用了4种微数据格式。一种用于评论本身（http://schema.org/Review），一种用于评论的事物（http://schema.org/Thing），一种用于评论者（http://schema.org/Person），还有一种用于评级系统（http://schema.org/Rating）。使用这些标准还可以标识出评论相关的更多细节。比如，可以添加一个餐馆的菜单、评论者的电子邮件地址，或者自定义评价系统的最小值和最大值。

在前述4种微数据格式中，通过`itemReviewed`使用http://schema.org/Thing与其他3种不同。乍一看，这里的Thing（事物）含义相当模糊。实际上作者是有意这么设计的，这样才能让你去标识任何可供评论的东西。而且，Thing也是一个基本词汇表，很多特殊的词汇表都以它为基础。比如产品、位置、活动、图书、音乐，等等（参见http://schema.org/docs/full.html）。在这个例子中，也可以不使用Thing而使用更具体的Restaurant（在http://schema.org/Restaurant中定义）。不过，由于这个页面并不包含有关餐馆的其他信息，所以没有必要。

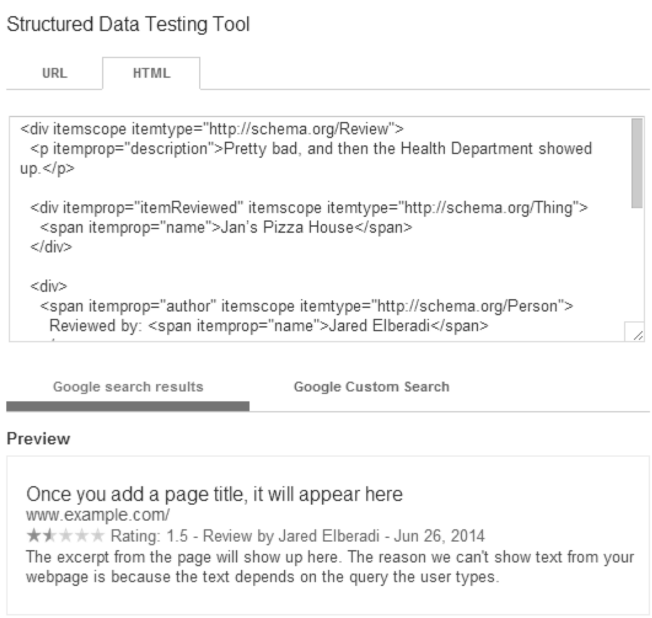


图3-9：搜索结果确实突出了评论，而且评价星级也非常抢眼

注意 可以参考微软的必应（Bing）如何使用Bing Markup Validator（http://www.bing.com/toolbox/markup-validator）处理元数据。如果你俄语很好，可以试试Yandex的微格式验证器，地址是http://webmaster.yandex.ru/microtest.xml。

怎么防止谷歌对你的语义数据视而不见

虽然谷歌能在搜索结果中对富含语义的页面给予特殊待遇，但这并不意味着它一定会这样做。谷歌自己有一套半保密的规则，这些规则决定了搜索用户是否能看到语义信息。不过，有一些做法肯定能让谷歌忽略你的数据。

- **语义数据不是主要内容**。换句话说，假如你在一个介绍飞蝇钓鱼的页面中硬加进去联系人信息，那谷歌就不太可能使用该信息。（想想看，搜索用户在找到这个页面时，他想知道的是有关钓鱼的信息，此时在页面标题下面显示你的地址和公司没有什么意义。）另一方面，假如你在自己的简历页面中加入联系人信息，那么该信息被采用的可能性就大得多。
- **语义数据被隐藏了**。谷歌不会使用通过CSS隐藏的语义数据。
- **你的网站中只包含很少的语义数据**。如果你的网站中只有为数不多的页面包含语义数据，谷歌很可能忽略这些页面。

避免了犯这些错误，就有机会让谷歌帮你显示更丰富的信息。

3.4.3 食谱搜索引擎

在搜索结果列表中显示更丰富的信息是一个不错的技巧，这样可以为网站带来更多流量。即便如此，你可能还是觉得不过瘾。这些刚刚掌握的技能还可以玩出什么花样来呢？告诉你吧，谷歌的天才工程师们一直在忙着设计未来的搜索，很多地方都考虑到了语义呢。

比如，有一个点子非常棒，那就是利用语义信息实现更智能的搜索过滤，而不是影响搜索结果的表现。假设人们使用RDFa、微格式或微数据标记了自己的简历，谷歌就可以提供一个特殊的简历搜索功能，专门检索这类数据，涵盖所有流行的求职网站，忽略其中无关的内容。这种简历搜索引擎也可以提供高级过滤选项，比如让公司可以按照认证或曾工作过的公司来搜索候选人。

目前，谷歌还没有发布这样的简历搜索引擎，但却发布了一个帮助有经验的专业人士找工作的技术（http://tinyurl.com/vetjobsearch）和一个产品搜索服务（http://www.google.ca/merchants）。但要说谷歌最成熟的基于元数据的搜索服务，还得说是它搜索菜谱的工具，可以搜索几百万种菜式。

说到怎样用微数据或微格式来标记食谱数据，恐怕已经难不倒你了。无非就是把一个食谱放到一个容器中，相应的格式遵守菜单数据格式（http://data-vocabulary.org/Recipe）即可。菜名、作者及图片都有各自的属性，而且还可以提供一句话的概述和来自用户的评论星级。

下面就是一个食谱的标记示例：

```
<div itemscope itemtype="http://data-vocabulary.org/Recipe">
  <h1 itemprop="name">Elegant Tomato Soup</h1>
  <img itemprop="photo" src="soup.jpg" alt="A bowl of tomato soup">
  <p>By <span itemprop="author">Michael Chiarello</span></p>
  <p itemprop="summary">Roasted tomatoes are the key to developing the rich
  flavor of this tomato soup.</p>
  ...
```

然后，可以再添加一些有关食谱的重要信息，比如准备时间、烹饪时间和菜量。此外，还可以再嵌套一个容器，用于包含营养方面的数据（相应的信息有份量、卡路里、脂肪，等等）。

```
...
<p>Prep time: <time itemprop="prepTime" datetime="PT30M">30 min</time></p>
<p>Cook time: <time itemprop="cookTime" datetime="PT1H">40 min</time></p>
<p>Yield: <span itemprop="yield">4 servings</span></p>
<div itemprop="nutrition" itemscope
  itemtype="http://data-vocabulary.org/Nutrition">
  Serving size: <span itemprop="servingSize">1 large bowl</span>
  Calories per serving: <span itemprop="calories">250</span>
  Fat per serving: <span itemprop="fat">3g</span>
</div>
...
```

注意 其中的prepTime和cookTime属性表示的是一段时间，而不是一个时间点。因此，不能使用与HTML5的<time>元素相同的时间格式。正确的格式是ISO格式，详情参见：http://tinyurl.com/ISOdurations。

再然后，就该列出食谱的各种原料了。每种原料都要单独放在一个嵌套的容器中，其中一般都要包含原料名和数量：

```
...
<ul>
  <li itemprop="ingredient" itemscope
   itemtype="http://data-vocabulary.org/RecipeIngredient">
   <span itemprop="amount">1</span>
   <span itemprop="name">yellow onion</span> (diced)
  </li>
  <li itemprop="ingredient" itemscope
   itemtype="http://data-vocabulary.org/RecipeIngredient">
   <span itemprop="amount">14-ounce can</span>
   <span itemprop="name">diced tomatoes</span>
  </li>
  ...
</ul>
...
```

原料部分的标记写起来确实比较烦琐——不过，别放弃，马上就能看到回报了。

最后，就是一系列实际操作步骤的详细说明，这些步骤要从属于一个属性，比如：

```
  ...
  <div itemprop="instructions">
    <ol>
      <li>Preheat oven to 450 degrees F.</li>
      <li>Strain the chopped canned tomatoes, reserving the juices.</li>
      ...
   </div>
  ...
</div>
```

要想看一个完整的食谱标记示例，请访问http://tinyurl.com/RichSnippetsRecipe。

注意 菜单一般都很长，也很琐碎，因此为它们加标记要花比较长的时间，还必须全身心投入。这种情况下，如果有一个得心应手的软件工具，显然能大大提高工作效率。想象一下，使用这个工具，作者只要在精心布局的窗口中的文本框中依次输入食谱的细节即可。然后，它就能生成语义正确、可以直接放到网页中的标记代码。

谷歌在索引了这个标记了食谱的页面后，用户就可以通过Recipe View（食谱视图）搜索到食谱。想试试Recipe View搜索？

(1) 请访问http://www.google.com/landing/recipes。

打开Recipe View之后，可以看到很多相关信息，还有视频教你如何使用。

(2) 单击“Try Google with Recipe View”按钮。

单击后会打开你熟悉的谷歌搜索页面，但有一些细微差别。在搜索框下面，可以看到红色的Recipes标签，表示你要搜索食谱。

(3) 然后在搜索框里输入菜名，单击Search。

谷歌一开始会搜索chicken pasta，你可以自己修改。

(4) 单击“Search tools”按钮（在搜索框右下方）。

因为谷歌能理解每个食谱的结构，从而可以忽略并不包含真正食谱数据的网页，而且还能添加更智能的过滤选项。单击“Search tools”按钮后，谷歌会显示三种筛选功能，以下拉列表形式呈现，恰好位于搜索结果上方（见图3-10）。

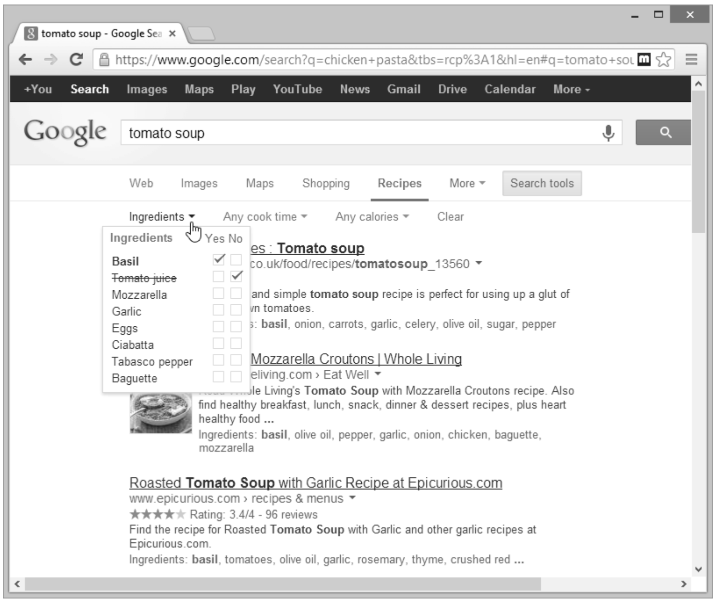


图3-10：搜索一个菜名之后，谷歌还可以让你通过匹配的食谱中包含的其他元数据来对结果进行过滤。比如，搜索一个西红柿汤食谱，其中用了大量罗勒，但没有用西红柿汁

- ❑ Ingredients（原料）。可以选择包含或不包含某种原料的菜式。只要单击相应原料旁边带Yes和No的小复选框即可。（为创建这个原料列表，谷歌要从你的搜索结果中取得最常用的那些原料。）
- ❑ Any cook time（烹饪时间）。可以找到最快捷的菜式，比如1小时或10分钟之内。
- ❑ Any calorles（卡路里）。适合节食的人，可以选择任意热量范围。

结果，语义数据为互联网用户提供了非常强大的信息挖掘支持，也为用户找到你的网页提供了更有效的方式。

第 4 章

构建更好的Web表单

HTML表单指的就是从网站访客那里收集信息的HTML控件。比如，可以填写文本的文本框，可以选择的列表框，可以选中或取消选中的复选框。在Web上，表单让人们可以做很多事情，从查询股票行情到购买演唱会门票。

HTML表单与HTML语言几乎是同时出现的，但到了20世纪末都没有丝毫改进。这期间的几次努力也以失败告终。制定Web标准的人们曾耗费数年时间推出了XForms，作为HTML表单的升级版。但XForms跟XHTML 2（参见1.1.2节）一样单调乏味。虽然XForms确实能解决一些问题，而且也不失方便和优雅，但它的问题也很多。比如，代码冗长，要求设计人员熟练掌握XML。不过，影响它推广的最大障碍还是不兼容HTML表单。换句话说，要使用XForms的话，开发人员就得心一横，眼一闭，把未来完全交给一个新的模型。这需要极大的勇气，同时也是理想主义的一种表现。最终，由于主流浏览器根本就没有实现XForms的打算（太复杂，实际使用率也不高），Web开发人员也就从来没有真正面临前述两难的抉择。

HTML5采取了完全不同的做法：在已有HTML表单模型基础上进行修订。在老版本的浏览器中使用HTML5表单也没有问题，只不过没有那么多增强而已。（这是件好事，因为IE10之前的版本不支持任何新的表单功能。）HTML5表单还新增了一些实用的表单特性，在此之前，这些特性需要开发人员引入一堆JavaScript代码或者一个JavaScript工具包才能实现。现在，HTML5让这些特性很容易获得。

本章就来学习所有HTML5表单的新功能。我们会介绍现有浏览器支持哪个功能，不支持哪个功能，而为了弥补现有的差异，可以使用什么过渡方案。此外，本章还会介绍在普通网页中嵌入富HTML编辑器的技术，但严格来讲，这项技术不是HTML5表单标准中规定的。

4.1　理解表单

当然，可能有的读者对表单并不陌生。不过，假如你对表单理解得没有那么深，本节可以帮你梳理一下有关表单的基本知识。

我们通常所说的Web表单，就是一组文本框、列表、按钮及其他可以点击的小控件，通过这些小控件可以收集网站访客的某些信息。表单在互联网上是随处可见的，可以通过表单注册电子邮件账户，发表对商品的评论或者在网上银行完成交易。谷歌的搜索表单恐怕要算是世界上最简

单的表单了——只有一个文本框（图4-1）。



图4-1：谷歌简练的搜索页面中有一个只包含基本元素的表单。使用这个表单跟使用其他任何表单没有区别——输入一些文本（在这里是要搜索的关键词），然后点击按钮提交该文本

所有基本表单的工作方式都类似，即用户填写信息然后单击按钮。此时，浏览器会收集用户输入的信息并将其发送给Web服务器。在服务器上，有软件程序负责处理信息，并决定下一步的操作。服务器端的这个程序可能要联系数据库，可能是读取数据也可能是写入数据，之后再把新页面发送给浏览器。

这里提到的负责处理通过表单提交的数据的服务器端程序可没有那么简单，处理数据的方式不下几百种。有些程序员愿意使用纯手工脚本操作原始的表单数据，而其他人可能会利用服务器端语言打包好的模块，直接处理封装在对象中的表单数据。但无论采用什么方式，过程都差不多：检查表单数据，对数据进行某种处理，然后再发回一个新网页。

注意　本书不会讨论任何服务器端的编程工具。实际上，服务器端使用什么工具都无所谓，因为你要使用的表单元素不会变，而这些元素要遵守的HTML5规则也不会变。

使用JavaScript绕过表单提交数据

有必要提醒大家一下，并不是只能使用表单向Web服务器发送用户输入的数据（虽然以前是这样）。如今，熟练的程序员都使用XMLHttpRequest对象（参见12.1.1节），在JavaScript代码中静悄悄地跟服务器通信。谷歌就是一个例子，它采用这种技术一方面取得搜索建议，就是显示在下拉列表中的建议条目；另一方面会在用户输入搜索关键词的过程中即时显示结果页面。当然，你得开启谷歌的即时搜索功能才行（www.google.com/instant）。

像Google Instant这样通过JavaScript完全绕过表单提交数据的情况并不少见。可以把这种技术当做一项功能来提供给用户，但不要把它作为必要功能。原因很简单，JavaScript并非无懈可击的（比如，在网速慢的情况下可能会有些奇怪的现象发生），而且还有一少部分人的浏览器不支持或关闭了JavaScript。

最后还要说一句，页面中包含表单，不一定非要把数据提交到服务器。相信你见过那些只用于执行简单计算的页面（比如抵押贷款利率计算器），这些页面中的表单不需要与服务器通信，完全依靠JavaScript完成计算并在页面中显示结果。

从HTML5的角度来看，无论是向服务器提交表单，还是在本地页面的JavaScript中直接使用数据，或者通过XMLHttpRequest对象将它传回服务器，真的都无所谓。无论如何，都需要使用标准的HTML表单控件来构建表单。

4.2　传统表单翻新

学习HTML5表单的最佳方式，莫过于找一个现有的例子，然后加以改进。图4-2展示了这样一个有待改进的例子。



图4-2：只要你上过网，就不难遇见与图中的“动物园管理员申请表”类似的表单，它负责从网页访客那里收集基本的信息

这个表单的标记没有什么新意。如果你以前编写过表单，在这里就不会看到任何新玩艺儿。首先，整个表单被包装在一个<form>元素里：

```
<form id="zooKeeperForm" action="processApplication.cgi">
  <p><i>Please complete the form. Mandatory fields are marked with
```

```
    a </i><em>*</em></p>
    ...
```

<form>元素用于组织所有表单部件（也称为控件或字段），负责告诉浏览器把数据提交到哪里，方法是在action属性中提供一个URL。假如你只想在客户端使用JavaScript操作表单，那么只要在action属性里指定一个井号（#）即可。

注意 HTML5新增了一种机制，支持把表单控件放在它所在的表单外面。方法是使用新的form属性引用表单的ID值（如form="zooForm"）。不过，如果浏览器没有实现这种机制，提交表单时就会忽略这些数据。换句话说，这个小的改进还不适合在真正的网页中应用。

像前面“动物园管理员申请表”这样精心设计的表单，都会使用<fieldset>元素划分几个逻辑块。每个块都有一个放在<legend>元素中的标题。以下是Contact Details部分的<fieldset>元素（其结果如图4-3所示）：

```
...
<fieldset>
  <legend>Contact Details</legend>
  <label for="name">Name <em>*</em></label>
  <input id="name"><br>
  <label for="telephone">Telephone</label>
  <input id="telephone"><br>
  <label for="email">Email <em>*</em></label>
  <input id="email"><br>
</fieldset>
...
```

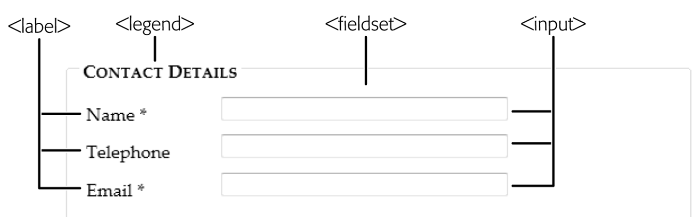


图4-3：这个<fieldset>元素收集三方面信息：名字、电话和电子邮件。每条信息都有一个文字说明（包含在<label>元素里）和一个负责收集数据的控件（使用<input>、<textarea>或<select>元素）

在任何表单里，通用的<input>元素都要承担大部分工作。通过<intput>可以收集文本，创建复选框、单选按钮和其他按钮。除了<input>元素，可以使用<textarea>元素让用户输入多行文本，而使用<select>元素则可以创建选择列表。想回顾这些表单元素的读者，可以参考表4-1。

表4-1　表单控件

控　件	HTML元素	说　明
单行文本框	`<input type="text">` `<input type="password">`	显示文本框，用户可以在其中填写内容。如果是密码类型的文本框，浏览器就不会显示用户输入的文本，而是用星号（*）或点号（·）来代替每个字符

（续）

控　　件	HTML元素	说　　明
多行文本框	<textarea>...</textarea>	显示大文本框，可以输入多行文本
复选框	<input type="checkbox">	显示复选框，可以作为开关，选中或取消选中
单选按钮	<input type="radio">	显示单选按钮（一个空心圆，可以选中或取消选中）。一般单选按钮都是成组出现的，每一组单选按钮都有相同的name属性，用户只能选择其中一个
按钮	<input type="submit"> <input type="image"> <input type="reset"> <input type="button">	显示标准的可以单击的按钮，其中类型为submit的提交按钮用于收集表单数据，并将它们发送给指定目标；类型为image的图像按钮与提交按钮作用相同，但可以显示成一幅可以单击的图像而非按钮；类型为reset的重置按钮，用于清除用户的选择和已经输入的文本信息；而类型为button的按钮本身没有任何功能，但可以通过JavaScript给它赋予功能
列表	<select>...</select>	显示一个选择列表，用户可以从中选择一或多个列表项。每个列表项用<option>元素添加

以下是“动物园管理员申请表”的剩余标记，包含了一些新元素（如<select>列表、复选框和提交表单的按钮）：

```
  ...
  <fieldset>
    <legend>Personal Information</legend>
      <label for="age"><em>*</em>Age</label>
      <input id="age"><br>
      <label for="gender">Gender</label>
      <select id="gender">
        <option value="female">Female</option>
        <option value="male">Male</option>
      </select><br>
      <label for="comments">When did you first know you wanted to be a
       zoo-keeper?</label>
      <textarea id="comments"></textarea>
  </fieldset>

  <fieldset>
    <legend>Pick Your Favorite Animals</legend>
    <label for="zebra"><input id="zebra" type="checkbox"> Zebra</label>
    <label for="cat"><input id="cat" type="checkbox"> Cat</label>
    <label for="anaconda"><input id="anaconda" type="checkbox"> Anaconda
    </label>
    <label for="human"><input id="human" type="checkbox"> Human</label>
    <label for="elephant"><input id="elephant" type="checkbox"> Elephant
    </label>
    <label for="wildebeest"><input id="wildebeest" type="checkbox">
     Wildebeest</label>
    <label for="pigeon"><input id="pigeon" type="checkbox"> Pigeon</label>
    <label for="crab"><input id="crab" type="checkbox"> Crab</label>
  </fieldset>
  <p><input type="submit" value="Submit Application"></p>
</form>
```

访问本书在线示例网站：http://prosetech.com/html5，可以看到完整的示例及简单的样式表。其中，ZookeperForm_Original.html是传统的没有经过改进的表单，而ZookeperForm_Revised.html包含的则是使用HTML5表单元素之后的新表单。

注意 HTML表单有一个限制，就是不能修改浏览器呈现控件的方式。例如，你想把标准的傻里傻气的灰色复选框替换成一个又大又醒目的黑白框，再配上粗大的对勾图标。对不起，HTML不支持。（解决方案是使用JavaScript来创建一个与复选框具有相同行为的常规元素，换句话说，就是在用户单击该元素时来回切换这个元素的外观。）

HTML5仍然没有打破这个不能自定义的限制，但新增了一些本章将会介绍的新控件。对于需要自由定制控件样式和希望统一页面外观的人，普通的HTML5表单不合适。相反，他们需要一个类似jQuery UI的JavaScript工具包。

看完了这个真正的表单示例之后，接下来我们就要动手用HTML5来改进它了。以下几节先从小的占位符文本和自动获得焦点的字段开始。

4.2.1 通过占位符文本添加提示

表单一开始通常都是空的，但一堆空空如也的文本框看起来会让人心里发慌，特别是在文本框归属关系不清的时候尤其如此。所以，我们经常也可以看到一些文本框里预先带有一段提示性文本。这种占位符文本也叫做水印，因为这些文本的颜色一般是浅灰色的，用以区别用户真正输入的文本。图4-4展示了占位符文本。

要创建占位符，使用placeholder属性：

```
<label for="name">Name <em>*</em></label>
<input id="name" placeholder="Jane Smith"><br>
<label for="telephone">Telephone</label>
<input id="telephone" placeholder="(xxx) xxx-xxxx"><br>
```

不支持placeholder属性的浏览器会忽略它；IE（IE10之前的版本）的可能性最大。好在占位符文本并非不可或缺，没有它也不会影响到表单的基本功能。假如你真想要占位符文本，实际上也有很多JavaScritp补丁可以让IE支持，参见http://tinyurl.com/polyfills。

目前，还没有标准的统一方式来改变占位符文本的样式（例如把它们变成斜体或换成其他颜色）。但浏览器开发商最终会提供一个应用CSS样式的挂钩——事实上，你在看本书的时候，可能他们正在商讨具体的实现方案呢。如果你现在就有需求，可以使用特定于浏览器的伪类（即Chrome浏览器的-webkit-input-placeholder、IE浏览器的-ms-input-placeholder和Firefox浏览器的-moz-placeholder）。附录A有关于伪类的详细介绍，6.1节介绍了特定浏览器样式的尴尬局面。



图4-4：上：如果文本框中没有内容，就显示占位符文本，如Name和Telephone字段所示；下：在用户单击文本框后（文本框获得焦点），占位符文本消失，当焦点转换到下一个文本框时，如果文本框仍然是空的，则占位符文本又会出现

话说回来，使用被支持得更好的focus伪类，可以在文本框获得焦点时轻松改变其样式。例如，可以让文本框的背景变成深色，以便更加显眼：

```
input:focus {
  background: #eaeaea;
}
```

用好占位符

并不是每个文本框都需要占位符，应该利用占位符来消除歧义。例如，“姓”这个文本框不用多加解释，而“名字”（如图4-4中的Name）就不是那么明确了。占位符文本告诉用户：名和姓之间要留一个空格。

在某些情况下，占位符则是一个示例值，即用户可能真的会输入的值。比如，微软的Hotmail登录页面（www.hotmail.com）使用文本someone@example.com作为占位符，说明你应该在输入框中输入你的邮箱地址，而不是你的名字或其他信息。

另外，占位符可以用来表示值的格式。图4-4中的电话号码占位符(xxx) xxx-xxxx表示电话号码应该由三位数字的区号开头，后跟三位，再跟四位数字。这种占位符不一定表示不能接受其他格式的输入，但却能够给用户一个格式方面的建议。

应该避免用占位符去做两件事儿。一是不要用它代替字段描述或说明。比如，对于一个收集信用卡安全码的文本框，“您的卡背面的三位数字”并不适合以占位符形式出现。可以考虑把它放在输入框下面，或者把这句话作为title属性的值，当用户鼠标悬停到字段上时，弹出一个窗口来告诉用户应该输入什么：

```
<label for="promoCode">Promotion Code
</label>
<input id="promoCode" placeholder="QRB001"
title="Your promotion code is three
letters followed by three numbers">
```

二是不要为了表示占位符不是真正的内容，就选择特殊字符作为占位符。比如，有些网站中使用[Jonh Smith]而不是John Simth，想用方括号强调这个占位符只是一个例子。这种做法很容易让人迷惑。

4.2.2 焦点：挑选正确的起点

加载完表单之后，用户要做的第一件事儿就是填写表单。然而，除非用户按下Tab切换到第一个控件，或者在其中单击一下鼠标，从而让第一个控件获得*焦点*，否则就不能输入。

在相应的<input>元素上通过JavaScript调用focus()方法，可以帮用户完成焦点切换。但这样就得编写代码，而且有时还会出现问题。比如，在调用focus()方法之前，用户已经单击了其他控件并开始输入的情况也可能发生。这时硬性地把焦点切换到第一个控件显得很粗鲁。但如果浏览器自己能控制焦点，它就可以在用户操作之前，先把焦点给予正确的控件。

这就是HTML5添加autofocus属性的初衷，但只能给一个<input>或<textarea>元素添加这个属性：

```
<label for="name">Name <em>*</em></label>
<input id="name" placeholder="Jane Smith" autofocus><br>
```

与placeholder属性一样，除了IE9及其更早的版本，其他浏览器都支持autofocus属性。同样，对于IE还是有办法的。可以使用Modernizr（参考1.6.3节）检测浏览器是否支持autofocus，然后自己编写脚本实现。也可以使用别人已经开发好的JavaScript程序（http://tinyurl.com/polyfills）。不过，很少有人为了这点功能如此兴师动众，除非你也想让IE支持其他表单功能，比如下面要讨论的数据验证。

4.3 验证：阻止错误

表单中的字段是为了从网页访客那里收集信息的。但是，无论你多么彬彬有礼地询问，都可能得不到想要的结果。性子急躁的或者糊里糊涂的访客，有可能跳过重要的部分，只填写少量信息，或者只是不小心按错了键。最后怎么样？他单击了“提交”按钮，你的网站收集到了一堆乱七八糟的数据。

很多网页都需要*验证*，就是在发生错误时捕捉到它（而更好的方案是防止出错）。很多年来，开发人员都要自己写JavaScript验证脚本，有时候也会用到专业的JavaScript库。应该说，这些验证方法效果都非常好。然而，鉴于验证是那么常用（可能要对每个人都做错误检查），验证解决的问题是那么集中（例如，发现无效的电子邮件或日期），以及编写验证脚本那么讨厌（没有人真心喜欢为每个表单都编写一遍同样的代码，更不要说*测试*这些脚本了），因此这个问题一定还有改进的空间。

HTML5规范的制定者决定让浏览器帮忙解决这些小问题，以后就不用让开发人员操心了。为此，他们设计了一套*客户端*验证方法（参见后面的“在两个地方验证”），让我们可以在<input>

字段里嵌入常用的错误检查规则。而且，嵌入验证规则的方法很简单，只要指定正确的属性就行。

4.3.1 HTML5 验证的原理

HTML5表单验证的基本原理就是你来告诉浏览器要验证哪个字段，但具体验证的细节，你不用管。这就像是提拔你当了领导，只不过没给你涨工资。

例如，你决定某个字段不能留空，意味着用户必须填上点什么。在HTML5中，可以通过`required`属性贯彻你这个指示：

```
<label for="name">Name <em>*</em></label>
<input id="name" placeholder="Jane Smith" autofocus required><br>
```


在两个地方验证

多少年来，敬业的开发人员已经找到很多解决验证问题的方法。今天，大家都有了明确的共识。那就是，要想让表单万无一失，就需要在两个地方验证。

- **客户端验证**。客户端验证就是在浏览器中检查错误，没有错误再提交。客户端验证的目的是减少填表人的麻烦。因为填表人要知道哪儿填得不对，不必填完30多个文本框再提交，而是随填随提示了。也就是说，可以在出错的字段旁边显示一条类似帮助的消息，提醒填表人在提交表单之前纠正错误。
- **服务器端验证**。这是在用户将数据提交给服务器之后进行的验证。此时，服务器端代码要负责检查所有细节，确保在进行下一步操作之前所有数据都是有效的。无论浏览器做不做验证，服务器端验证都是必不可少的。这是预防别有用心的人故意篡改数据的唯一途径。如果服务器端验证检测到问题，就向浏览器发回一个包含错误消息的页面。

简言之，客户端验证（包括HTML5表单验证）是为访客提供方便的，而服务器端验证才是确保数据正确性的。最关键的是，要知道这两个地方的验证缺一不可——除非你的表单极其简单，而且不担心数据有错误或者有错误问题也不大。

一开始并没有什么可见的提示告诉用户这是一个必填字段。为此，我们应该给出一些提示，比如为文本框应用不同的边框颜色或者在字段旁边放一个星号（就像前面“动物园管理员申请表”一样）。

当填表人单击提交按钮时，验证才发挥作用。如果浏览器支持HTML5表单，当它发现有一个必填字段为空时，它会拦截提交，并在无效字段旁边显示一个提示，如图4-5所示。

下面几节会介绍，不同的属性表示不同的验证规则。可以给一个输入框应用多种规则，而一种规则可以应用给多个`<input>`元素（或者`<textarea>`元素）。提交表单之前，所有验证条件都必须满足。

而这就引发了一个有意思的问题：如果表单数据违反了多个规则会出现什么结果——比如有多个必填字段都空着？

同样，在用户填写完表单并单击提交之前，什么都不会发生。只有单击“提交”按钮，才会

触发浏览器去从上到下地验证表单数据。只要发现一个无效的值，它就会停下来，不再继续验证其他字段。同时，它会取消提交操作，并在无效值的旁边显示一条错误消息。（此外，如果有问题的文本框不在当前视口中，则浏览器会滚动到它正好位于页面顶部。）用户纠正了输入错误并再次单击“提交”按钮后，浏览器会在下一个无效的值处停下来，再次给出错误提示。

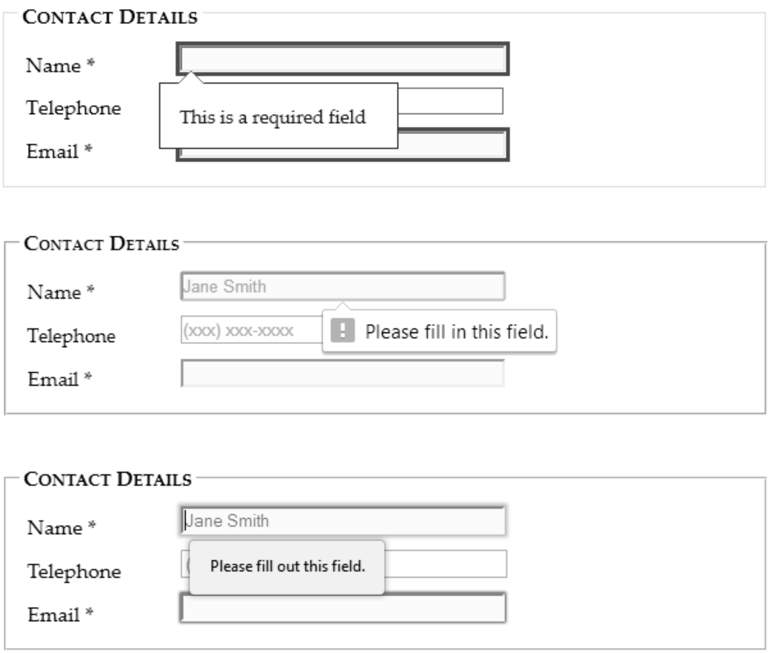


图4-5：这里是相同的必填字段在Chrome（上）、IE（中）和Firefox（下）中的样式。浏览器采用什么方式提醒用户，在规范里没有限制，但这三个浏览器都使用了类似提示条的弹出框。可惜的是，我们不能修改这个弹出框的样式，也不能修改验证消息——至少现在不能

注意 只有用户单击“提交”按钮，浏览器才会执行验证。这样可保证验证的效率，同时也比较适度，因而所有人都可以使用。

有些人喜欢在用户输入错误并离开（当他们按Tab切换焦点或单击网页中其他地方）相应字段时马上给出提示。对于比较长的表单，特别是用户可能会在多个不同字段中犯相同错误的情况下，这种方法很有必要。遗憾的是，HTML5不支持指定验证的时机，但将来倒是有这个可能。目前，如果想要即时显示验证消息，最好还是自己编写JavaScript或选择一个不错的JavaScript库。

4.3.2 关闭验证

有些情况下，可能需要禁用验证功能。比如，在测试服务器端代码能否适当处理无效数据的时候，就需要关闭客户端验证。要禁用整个表单的验证功能，可以在`<form>`元素中添加`novalidate`属性：

```
<form id="zooKeeperForm" action="processApplication.cgi" novalidate>
```

另外，也可以考虑增加另外一个提交按钮来绕过验证。这个办法有时候对网页是很有用的。比如，可以给出一个正式的提交按钮，强制对表单进行严格验证，而同时再给出另一个提交按钮，实现其他功能（如保存未完成的数据，以便将来使用）。要添加这个额外的按钮，可以在表示相应按钮的<input>元素中添加formnovalidate属性：

```
<input type="submit" value="Save for Later" formnovalidate>
```

好了，现在已经介绍了怎么通过验证来捕获缺失的信息。接下来，我们讨论如何查找不同类型数据中的错误。

注意 什么，你想验证数值？唉，没有验证规则强制文本中必须包含数字——不过，倒是有一个新的number数据类型，4.4.5节会讨论到。可惜的是，浏览器对这个新数据类型的支持还不好。

4

4.3.3 验证样式挂钩

虽然我们无法修改验证消息的样式，但却可以根据输入字段是否已验证来改变它们的外观。比如，在输入的值无效时可以换一种背景颜色，只要浏览器一检测到问题，文本框的背景颜色就会立刻改变。

为此，只要使用几个新的伪类即可（关于伪类，请参考附录A），可以使用的伪类如下。

- ❑ **required**（必填）和**optional**（选填）：根据字段中是否使用了required属性来应用不同的样式。
- ❑ **valid**（有效）和**invalid**（无效）：根据控件中是否包含错误来应用不同的样式。注意，除非访客提交表单，否则大多数浏览器并不会发现哪些值有效，哪些值无效；换句话说，访客不会实时看到表示输入无效的样式变化。
- ❑ **in-range**（在范围内）和**out-of-range**（超出范围）：根据控件的min和max属性判断输入值是否超出范围，从而为控件应用样式（参见4.5.5节）。

举个例子，假如你想为一个必填的<input>元素应用浅黄色背景，就可以为required伪类定义一条样式规则：

```
input:required {
  background-color: lightyellow;
}
```

或者，如果你只想突出显示那些必填且当前填入了无效值的字段，那么可以像下面这样把required和invalid伪类组合起来：

```
input:required:invalid {
  background-color: lightyellow;
}
```

有了这条规则，空着的字段就会自动高亮，因为这些字段违反了必填字段的规则。

可以灵活运用上述技巧，比如组合valid伪类与focus伪类，或者为表示无效的值使用一个带错误图标的偏移背景。当然，还有一个忠告：可以使用这些伪类增强页面体验，但要保证页面没有它们也照样很好——旧版本的浏览器不支持！

4.3.4 使用正则表达式

HTML5支持的最强大（也最复杂）的验证方法是正则表达式。既然JavaScript支持正则表达式，那么为HTML表单添加这项功能也在情理之中。

所谓*正则表达式*，就是一种用正则表达式语言编写的文本模式。正则表达式的用途是匹配文本——比如，可以用正则表达式验证邮政编码中包含正确的字母和位数，或者验证电子邮件地址中包含一个@符号和一个至少两个字母的域名后缀。还是看一个正则表达式吧：

```
[A-Z]{3}-[0-9]{3}
```

开头的方括号定义了允许的字符范围。换句话说，[A-Z]就是允许从A到Z的任意字母。随后的花括号表示允许几个字符，{3}当然表示允许3个大写字母。接下来的短横线没有特殊的含义，就表示三个大写字母后面必须有一个短横线。最后，[0-9]表示允许一个0到9的数字，而{3}要求必须是3个数字。

正则表达式经常用于搜索（在长文档中查找匹配模式的文本）和验证（验证某个值匹配模式）。HTML5表单使用正则表达式来验证。

注意 正则表达式极客请听好——不必使用^和$字符表示要匹配字段值的开头和结尾。HTML5会自动确保这一点。实际上，这就是说正则表达式匹配的是字段中**完整**的值，验证的也是整个值的有效性。

好，下面这些值都是有效的，因为它们与上面的模式匹配：

```
QRB-001
TTT-952
LAA-000
```

但下面这些值无效：

```
qrb-001
TTT-0952
LA5-000
```

常用的正则表达式很可能比这个例子所展示的复杂。而且，编写正则表达式本身也很烦琐。正因为如此，大多数开发人员都愿意搜索一个现成的正则表达式来验证相关的数据类型。要不然，也会找别人帮忙。

提示　要想轻松学习正则表达式语言的基本规则，能够编写简单的表达式，可以参考以下简明教程：http://www.w3schools.com/js/js_obj_regexp.asp或http://tinyurl.com/jsregex。要想找一些现成的正则表达式，用在自己的表单里，可以访问http://regexlib.com/。要想成为一名正则表达式高手，可以看《精通正则表达式》(Jeffrey Friedl著，O'Reilly英文版；余晟译，电子工业出版社中文版)。

找到或写好一个正则表达式之后，可以通过pattern属性将其应用到<input>或<textarea>元素：

```
<label for="promoCode">Promotion Code</label>
<input id="promoCode" placeholder="QRB-001" title=
 "Your promotion code is three uppercase letters, a dash, then three numbers"
 pattern="[A-Z]{3}-[0-9]{3}">
```


图4-6展示了输入的值违反正则表达式规则后的结果。

提示　浏览器不会验证空值。在这个例子中，不输入折扣号（Promotion Code）可以通过验证。如果这不是你希望的结果，那么就要同时指定pattern和required属性。

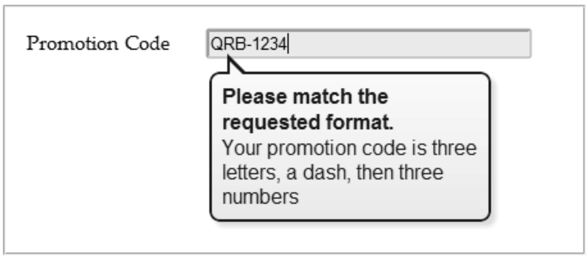


图4-6：聪明的浏览器(如这里的Chrome)不仅能捕获错误，还能从title属性中提取文本并显示出来，以告知填表的人

注意　正则表达式似乎能完美地匹配电子邮件地址(确实也是)。尽管如此，建议大家也不要用正则表达式，因为HTML5专门定义了一个用于输入电子邮件地址的输入类型，当然已经内置了正确的正则表达式(具体请参见4.5.1节)。

4.3.5　自定义验证

HTML5规范也规定了一组JavaScript属性，通过它们可以知道字段是否有效(或强制浏览器验证这些字段)。其中最常用的是setCustomValidity()方法，基于这个方法可以针对特定字段编写自定义的验证逻辑，并利用HTML5的验证机制。

下面来看看怎么自定义验证。首先，要检查相应字段是否有错，为此需要处理onInput事件，没有什么要解释的：

```
<label for="comments">When did you first know you wanted to be a
 zookeeper?</label>
<textarea id="comments" oninput="validateComments(this)" ></textarea>
```

这里，onInput事件会触发一个名为validateComments()的函数。这个函数的代码是你自己写的，主要是检测<input>元素的值，然后调用setCustomValidity()方法。

如果当前值有问题，那么在调用setCustomValidity()方法时就需要提供一条错误消息。否则，如果当前值没有问题，调用setCustomValidity()方法时只要传入空字符串即可；这样会清除以前设置过的自定义错误消息。

要强制评论（comment）框中至少有20个字符，可以这样写validateComments()函数：

```
function validateComments(input) {
  if (input.value.length < 20) {
    input.setCustomValidity("You need to comment in more detail.");
  }
  else {
    //没有错误。清除任何错误消息
    input.setCustomValidity("");
  }
}
```

图4-7展示了违反上述规则并提交表单的结果。



图4-7：如果在调用setCustomValidity()方法时提供了错误消息，浏览器会将该消息当做自己内置的消息。提交表单时，就会看到弹出的警告框中包含自定义的错误消息

当然，对于验证需要长字符串的字段，使用正则表达式可能更简单明了。而尽管正则表达式很适合验证某些数据类型，自定义验证逻辑却适用于任何情况，无论是复杂的数学计算还是与Web服务器通信。

注意 网页访客可以看到你放在JavaScript中的一切，这里没有什么加密算法。就拿前面示例中的折扣码来说，其位数是12位。但你肯定不愿意在自定义验证逻辑中透露这个信息，否则无疑会为那些想要伪造折扣码的人提供便利。对于这种情况，还是把验证逻辑放在服务器端比较好。

4.4 浏览器对 Web 表单和验证的支持

不同浏览器对HTML5表单的支持也不一样。换句话说，有的浏览器支持某些验证功能，而另一个浏览器则未必支持同样的功能。表4-2列出了支持到目前为止介绍的所有验证功能的浏览器及它们的最低版本。正如下表所表明的那样，有两个令人头疼的支持问题：旧版的IE浏览器和运行在智能手机和平板电脑上的移动浏览器。

表4-2　支持验证功能的浏览器

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android版的Chrome
最低版本	10	4	10	5*	10	–	28

*Safari不支持所要求的属性。

由于HTML5验证不能取代服务器端验证，因此可以仅将其看做一种增强；在这种情况下，浏览器支持的一致不一致也就无所谓了。此时，对于不支持这些验证的浏览器（如IE9），用户提交的数据会直接发送给服务器，然后服务器发现其中的问题，再返回带错误消息的相同页面。

另一种可能是，你的网站包含一些复杂的表单，有些到底该怎么填也不太明确，而你也不愿意眼睁睁看着大量IE用户流失。此时，有两个选择：你自己编写验证功能或使用现成的JavaScript库，借助外脑。到底怎么办，取决于验证任务的广度和复杂性。

4.4.1 用Modernizr检测支持情况

假如你的表单只需少量简单的验证，你可以自己写验证代码。利用Modernizr（参见1.6.3节）可以检测浏览器对各种HTML5表单验证功能的支持情况。例如，使用`Modernizr.input.pattern`属性可以检测出浏览器是否支持`pattern`属性：

```
if (!Modernizr.input.pattern) {
  //浏览器不支持正则表达式验证，可以在JavaScript中使用正则表达式
  ...
}
```

注意　这里的`pattern`属性只是`Modernizr.input`对象中的一个属性。其中，用于检测浏览器对表单验证支持情况的属性还有：`placeholder`、`autofocus`、`required`、`max`、`min`和`step`。

没错，这个例子没有告诉你什么时候执行检测，也没有告诉你该如何反馈。假如你想模仿HTML5验证机制，可以在用户提交表单的时候执行检测。为此，需要为表单的`onSubmit`事件定义处理函数，根据情况返回`true`（表示验证通过，可以提交表单）或`false`（表示验证未通过，浏览器应该取消提交操作）：

```
<form id="zooKeeperForm" action="processApplication.cgi"
 onsubmit="return validateForm()">
```

以下是一个简单的示例，演示了针对一个必填字段的自定义验证代码：

```
function validateForm() {
  if (!Modernizr.input.required) {
    //不支持required属性，因此必须自己编写代码检测

    //首先，取得包含所有元素的数组
    var inputElements = document.getElementById("zooKeeperForm").elements;

    //接着，遍历数组，检测每个元素
    for(var i = 0; i < inputElements.length; i++) {

      //检测当前元素是否必填
      if (inputElements[i].hasAttribute("required")) {
        //如果是必须填写的，则检测其值是否为空
        //如果为空，则表单验证失败，返回false
        if (inputElements[i].value == "") return false;
      }
    }

    //如果到了这儿，则一切顺利
    //浏览器可以提交表单
    return true;
  }
}
```

提示 以上代码运用了基本的JavaScript技术，包括查找元素、循环和条件逻辑。有关JavaScript的基本知识，请参阅附录B。

4.4.2 用HTML5Forms兼容

如果你的表单很复杂，而你又想省点事儿（省下的时间可以学习未来的技术），则大可选择一个现成的JavaScript库。从技术角度讲，无论是自己写，还是使用JavaScript库都没有什么不一样。都是先检测浏览器对验证功能的支持情况，然后再在必要时手工验证。但区别在于，JavaScript库已经为你准备好了所有代码。

访问http://tinyurl.com/polyfills，可以看到长长的页面，里面列出了大量JavaScript库。其中一个古老但好用的HTML5Forms库，可以从http://tinyurl.com/html5forms中获取。要下载一份，点击Download ZIP按钮，会获得一个打包了所有文件的Zip包。解压它，会得到一堆有用的脚本（在shared/js文件夹）和一长串示例页面（在tests/html5forms文件夹）。

要使用HTML5Forms库，需要把shared文件夹（包括所有子文件夹）复制到你网站所在的文件夹中。你可以给它重命名（比如，将shared改名为html5forms），只要同时修改脚本引用里的名字。一旦复制了这些文件，需要在网页中添加两个引用，像这样：

```
<head>
  <title>...</title>
  <script src="shared/js/modernizr.com/Modernizr-2.5.3.forms.js"></script>
  <script src="shared/js/html5Forms.js" data-webforms2-support="all"
   data-webforms2-force-js-validation="true">
  </script>
  ...
<head>
```

第一个引用指向了一个包含在HTML5Forms中的Modernizr的小版本（这里，它的名字是Modernizr-2.5.3.forms.js），并且提供了特性检测，以确保验证模块仅在浏览器需要的时候加载。如果已经在使用Modernizr，应该忽略这个引用，只需确保Modernizr版本包含了表单检测模块。这些是在Modernizr下载页面“Non-core detects”部分以`forms-`（比如`forms-validation`）开头的模块。

4

第二个引用指向HTML5Forms库。在熟悉的`src`属性后面，会看到一个或多个指定所需功能的属性。在上面的例子中，脚本加载了所有的`webforms`功能。HTML5Forms是一个模块化的库，这意味着可以选择性地使用它的一部分功能。这种策略确保了页面不会费神加载不需要的功能。下面的例子中选择了支持基本的验证，必需的字段和占位符：

```
<script src="shared/js/html5Forms.js"
 data-webforms2-support="validation,placeholder"
data-webforms2-force-js-validation="true">
```

提示　如果想要一个不同的功能组合，可以在下载的HTML5Forms的tests/html5forms目录下，找一个对应的示例页面文件。

接下来会介绍HTML5Forms库支持得非常好的一些表单特性，比如滑动条、日期选择器和选色器。代码中仍然有一些坑和少量的bug。如果你计划使用这些新控件，用之前最好在旧浏览器（比如IE9）上测试一下。

几个特殊的输入属性

HTML5还定义了另外几个不用于验证的属性，而是用于在编辑表单时控制浏览器的行为。这些属性并不是所有浏览器都支持。因此，这几个属性目前还只能用于试验。

- **Spellcheck**。有些浏览器可以帮用户检查输入的拼写是否正确。不过有一个明显的问题，即并非所有输入都是单词，而这个功能只允许用户“胡乱”输入几个字母。将`spellcheck`设置为`false`，表示不建议浏览器对字段进行拼写检查；设置为`true`，表示建议拼写检查。（浏览器默认的拼写检查行为也不一样，如果你不设置`spellcheck`属性，那么这个问题就会出现。）
- **Autocomplete**。有些浏览器为了节省你的时间，会在你向字段中输入信息时提供最近输入的值供你选择。自动完成功能并不是所有时候都适用的，正如HTML5规范中指

出的，有些信息属于敏感信息（比如，核攻击的编码），而有些信息只是临时性的（如一次性的银行登录验证码）。在这种情况下，应该把autocomplete属性设置为off，告诉浏览器不要提供自动完成的建议。而在确实需要的情况下，也可以把autocomplete设置为on。

- **Autocorrect**和**autocapitalize**。这两个属性可以用来在移动设备（即iPad和iPhone中的Safari）上控制自动纠错和自动大小写功能。
- **Multiple**。很久以来，Web开发人员一直通过为<select>元素添加multiple属性，达到让用户能选择多个列表项的目的。但现在，可以为某些类型的<input>元素添加这个属性，包括用于上传文件的file类型和email类型（参见4.4.1节）。在支持的浏览器中，用户可以选择多个文件一块上传，或者可以在一个输入框中贴上多个邮件地址。

4.5 新的输入控件

HTML表单有一个奇怪的做法，即用一个元素（含含糊糊地叫<input>）创建多个控件：复选框、文本框，以及按钮。此时，type属性就成为地地道道的总开关，它的值决定了<input>元素到底是什么控件。

如果浏览器遇到了不认识的<input>元素类型（type属性值），它就将其作为一个普通的文本框来处理。换句话说，以下3个元素在浏览器中都会生成一个文本框：

```
<input type="text">
<input type="super-strange-wonky-input-type">
<input>
```

HTML5利用了浏览器的这个默认处理方式，为<input>元素添加了新的类型，如果浏览器不认识这些类型，仍然会将其当做普通的文本框来处理。比如，要创建一个用于输入电子邮件地址的文本框，可以使用新的email类型：

```
<label for="email">Email <em>*</em></label>
<input id="email" type="email"><br>
```

在不支持email类型的浏览器（如IE9）中打开这个网页，会看到一个普通文本框，这是完全可以接受的。但支持HTML5表单的浏览器会更聪明一点，它们会像下面这样做。

- **提供便于编辑的辅助**。例如，智能一些的浏览器或者便捷的JavaScript组件可以从你的地址簿中取得电子邮件地址，帮你填到电子邮件字段中。
- **限制可能出现的错误**。例如，在数值文本框中输入的字母会被浏览器忽略，或者无效的日期会被拒绝（当然，也可能会要求你从一个迷你小日历中选择日期，这样既方便又可靠）。
- **执行验证**。在单击提交按钮时，浏览器可以执行更加完善的检查。比如，智能一些的浏览器会发现电子邮件字段中明显存在错误的邮件地址，从而拒绝继续提交。

HTML5规范没有就上述第一点作出明确规定。浏览器开发商可以视情况自行决定如何显示和编辑不同的数据，而不同的浏览器也可以有自己的特色功能。比如，移动设备上的浏览器可以

定制虚拟键盘，显示或隐藏不需要的键（参见图4-8）。



图4-8：在移动浏览器中，要填写表单可没有全键盘可以使用。图中的iPod通过定制虚拟键盘为用户提供了方便，根据要输入的数据类型——电话号码和电子邮件，会分别显示数字键盘（左）和带有@按键及小空格键的字母键盘（右）

不过，更重要的还是预防错误和检测错误的功能。最起码，支持HTML5表单的浏览器在发现表单中包含违反数据规则的数据时，要阻止表单提交。所以，如果浏览器不能做到预防错误（即上述第二点），那它必须在用户提交表单时验证数据（上述第三点）。

然而，目前并非所有浏览器都达到了这些要求。有的支持新控件类型，也提供了一些编辑辅助，但缺少验证功能。而很多只支持其中部分新控件，且不同浏览器支持的范围又不一样。移动浏览器的问题最多，虽然它们提供编辑辅助功能，但却不会对数据进行验证。

表4-3列出了新的控件类型以及完全支持它们的浏览器——完全支持是指在有数据违反规则时，浏览器会阻止表单提交。

表4-3　支持新控件的浏览器

控件类型	IE	Firefox	Chrome	Safari	Opera
email	10	4	10	5	10.6
url	10	4	10	5	10.6
search*	n/a	n/a	n/a	n/a	n/a
tel*	n/a	n/a	n/a	n/a	n/a
number	10	–	10	5	9
range	10	23	6	5	11
date、month、week、time	–	–	10	–	11
color	–	–	20	–	–**

*　HTML5规范没有要求验证这种类型的数据。

** Opera在版本11和12上支持了颜色控件类型，但在更新的版本中移除了这项支持。

> **提示** 如果你使用Modernizr，可以检测`Modernizr.inputtypes`对象的属性来确定浏览器支持的控件类型。比如，如果`Modernizr.inputtypes.range`返回`true`，则说明浏览器支持`range`类型的控件。

4.5.1 电子邮件地址

电子邮件地址使用`email`类型。一般来说，有效的电子邮件地址是一个字符串（当然，有些字符是不允许出现的）。这个字符串中必须包含@符号和一个点号，而且两者之间至少要间隔一个字符，点号后面至少也要有两个字符。以上差不多就是一个有效电子邮件地址的验证规则了。可是，要为验证电子邮件地址编写代码或者正则表达式，就没有这么简单了。这件看似简单的任务难倒过很多善意的开发人员。因此，最好还是找一个支持`email`控件的浏览器，让它自动帮我们检测得了（参见图4-9）。

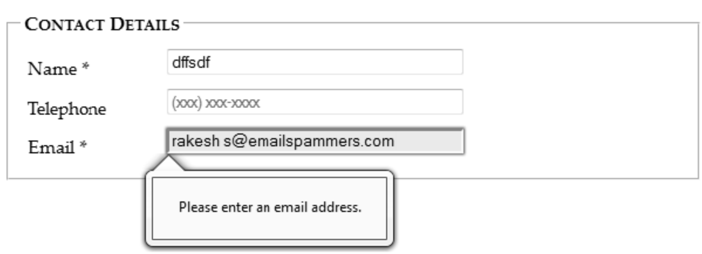


图4-9：Firefox拒绝接受伪造电子邮件地址中的空格

电子邮件控件支持`multiple`属性，添加了这个属性后就可以在同一个字段里输入多个电子邮件地址。不过，多个电子邮件地址之间只有逗号分隔，看起来仍然像一个字符串。

> **注意** 再提醒一次，空值可以通过验证。如果你想强制用户必须输入有效的电子邮件地址，就要在`email`控件中指定`required`属性（参见4.3.1节）。

4.5.2 网址

网址使用`url`类型。说到网址是由什么组成的，可能会引发激烈的讨论。但大多数浏览器都会对验证网址采用粗略的算法。首先要有一个URL前缀（可以是合法的，如http://；也可以是编造的，如bonk://），然后可以是空格和大多数特殊字符（冒号除外）。

有些浏览器也会在网址控件中给出URL建议，这些建议项一般是从浏览器最近的历史记录中提取的。

4.5.3 搜索框

搜索框使用search类型。搜索框中通常要输入关键词，用于执行某种搜索。可能是搜索整个互联网（比如使用谷歌，如图4-1所示），也可能是搜索一个网页或者对自己的某些信息执行定制搜索。无论如何，搜索框的样子与行为都与常规的文本框没有太大区别。

在Safari等浏览器中，搜索框的样式可能会稍有不同——两端都是圆形。而在Safari和Chrome的搜索框中输入关键词时，一个X图标就会立刻出现在搜索框的右侧，单击就可以清除搜索框。除了这些细微的差别之外，搜索框与文本框无异。但搜索框的值是有其特定语义的。换句话说，使用搜索框可以让浏览器及辅助（残障人士）上网的软件知道它是干什么用的。也许将来会有一天，这些工具能够利用搜索框把访客引导到正确的位置，或者为用户提供一些便利功能。

4

4.5.4 电话号码

电话号码使用tel类型。电话号码有很多种模式，有的只包含数字，有的还会包含空格、短横线、加号和圆括号。正是因为存在这么多差异，HTML5规范没有要求浏览器验证电话号码。不过，谁都知道电话号码字段至少不能接受字母（当然，tel控件确实不接受字母）。

目前，使用tel类型控件的唯一用途是在移动浏览器中定制虚拟键盘，键盘中只包含数字，没有字母。

4.5.5 数值

HTML5定义了两种数值类型的控件。其中，number类型用于常规数值。

使用number类型的控件有明显的好处。常规文本框什么值都可以接受：数值、字母、空格、标点符号，以及一些专门的卡通的字符。为此，检测输入的值是不是数值以及是不是在某个范围内就成了非常重要的任务。现在有了number类型的控件，浏览器就可以自动忽略非数值字符。看一个例子吧：

```
<label for="age">Age<em>*</em></label>
<input id="age" type="number"><br>
```

当然，数值也有很多种，也并非任何数据形式都可以接受任意数值。上面标记中所示的年龄（age）可以接受43 000、–6之类的值。为了增加限制，需要配合使用min和max属性。比如，下面的例子就把可接受的数值限制在了0到120之间：

```
<input id="age" type="number" min="0" max="120"><br>
```

一般来说，number控件只接受整数，不接受30.5这样的小数。（实际上，有些浏览器都不允许输入小数点。）不过，通过设置step属性可以改变这一点；step属性表示可以接受的数值之间的间隔。例如，将最小值（min）设置为0，将间隔值（step）设置为0.1，意味着可以输入0、0.1、0.2、0.3……然而，输入0.15后提交表单就会收到错误消息。默认的间隔值为1。

```
<label for="weight">Weight (in pounds)</label>
<input id="weight" type="number" min="50" max="1000"
step="0.1"value="160"><br>
```

设置step属性也会影响到数值框的微调按钮，如图4-10所示。



图4-10：很多浏览器都会在数值框中添加微调按钮。每单击一次向上箭头，数值都会增加step属性指定的值（除非已经达到允许的最大值）。类似地，每单击一次向下箭头，数值都会减少step属性指定的值

4.5.6 滑动条

HTML5的另一个数值类型的控件是range。与number控件类似，它也可以表示整数或者小数值。同样，range控件也支持与number控件相同的属性（min和max），用于设置允许的范围。下面是一个例子：

```
<label for="weight">Weight (in pounds)</label>
<input id="weight" type="range" min="50" max="1000" value="160"><br>
```

两者的区别是range控件用滑动条的形式表示信息。智能浏览器对于range控件，会显示一个如图4-11所示的滑动条，而不是文本框。

要设置range控件的值，只要把滑块拖动到合适位置即可，也就是把滑块放在滑动条上最大值和最小值之间的某个地方。支持range控件的浏览器不会告诉你最终设定了多大的值。如果你想显示这个值，可以使用JavaScript响应滑动条的变化事件（即处理onChange事件），然后在旁边把值显示出来。当然，你得事先检测一下浏览器是不是支持range控件（使用Modernizr等工具）。如果浏览器不支持range控件，就没有必要多此一举了，因为结果只能是在文本框中输入值。



图4-11：这个range控件与我们熟悉的音量调节器很像，它非常适合在最小值和最大值已知、范围适中且输入的特定值并不重要（但该值接近最小值还是最大值重要）的情况下使用

4.5.7 日期和时间

HTML5定义了几个与日期有关的新控件。支持日期控件的浏览器会提供一个方便的下拉式日历，供用户选择。这样，不仅可以避免对日期格式的困惑，也可以避免意外（或有意）输入一个不存在的日期。智能的浏览器还能提供更多便利，比如与个人日历集成。

目前来看，虽然日期控件很有用，但支持浏览器对它的支持还不好。Chrome和Opera是仅有

的提供下拉式日历的浏览器（如图4-12所示）。其他浏览器不支持日期数据类型，显示一个普通的、不可验证的文本输入框。

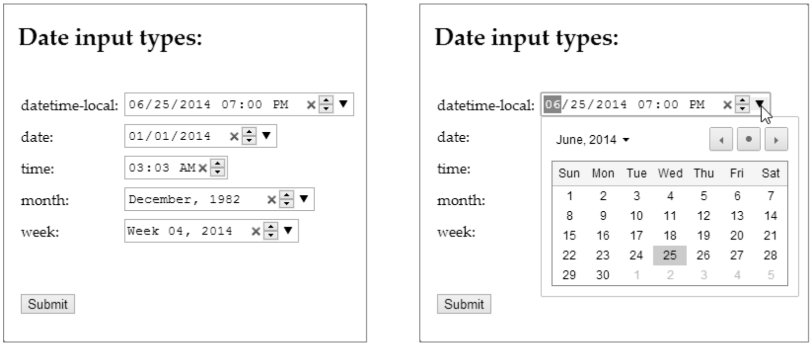


图4-12：date和time控件的文本框（左）有些不一样。但真正方便的地方在于，支持的浏览器提供了下拉式日历，让用户可以选择适当日期，而不必担心格式（右）

提示　假如你想使用新的日期控件，最好还是在不支持的浏览器中使用JavaScript库，例如前面介绍的HTML5Forms（4.4.2节）。毕竟，在不支持这些新日期控件的浏览器中，用户很容易输入格式错误的日期，而验证这些值和提供适当的说明也是很麻烦的事儿。（这就是为什么早就有了自定义JavaScript日期控件，而且它们在互联网上随处可见的原因。）

表4-4中列出了6种新的日期时间型控件。

表4-4　日期时间类型的控件

控件类型	说　明	示　例
date	格式为*YYYY-MM-DD*的日期	*2014-01-25*表示2014年1月25日
time	格式为*HH:mm:ss.ss*，用24小时制表示的时间，秒的部分可选	*14:35*或*14:35:50.2*表示下午2:35（50.2秒）
datetime-local	格式为*YYYY-MM-DDTHH:mm:ss*，包含日期和时间，中间以大写T分隔	*2014-01-25T14:35*表示2014年1月25日下午2:35
datetime	格式为*YYYY-MM-DD HH:mm:ss-HH:mm*，包含日期和时间，还有一个时区偏移量；与3.1.1节中的<time>元素格式相同。但是，datetime格式没有被任何浏览器有效支持，并且将来可能会移除，所以请使用datetime-local代替	*2014-01-15T14:35-05:00*表示美国纽约（西五区）时间2014年1月15日下午2:35
month	格式为*YYYY-MM*，表示年月	*2014-01*表示2014年1月
week	格式为*YYYY-Www*，表示年和周；根据年份不同，一年可能有52周或53周	*2014-W02*表示2014年第二周

提示　支持日期类型的浏览器也支持min和max属性。换句话说，可以设置最小和最大日期，但日期格式必须正确。比如，要限定某日期字段中必须填写2014年的日期，可以这样写：`<input type="date" min="2014-01-01" max="2014-12-31">`。

4.5.8 颜色

颜色使用color控件类型。虽然用处不大，但这个新控件很有意思，可以让用户从下拉式色盘中选取颜色，这个色盘就像在桌面绘图程序中看到的一样。目前，Chrome是唯一提供下拉色盘的浏览器。Oprea在版本10和版本11中曾经短暂地支持过，但是在认为太过试验性后就移除了该项支持。

在不支持color类型的的浏览器中，填写表单的用户需要靠自己输入一个十六进制的颜色码（或者可以使用4.4.2节介绍的HTML5Forms库）。

4.6 新元素

迄今为止，我们介绍了HTML5对表单的扩展，介绍了新的验证功能，也介绍了通过添加新的控件让表单更加智能。这些新功能都是很实用的，也得到了广泛支持。但是，这些并不是HTML5表单的全部。

HTML5也新增了一些全新的元素，用于弥补缺漏和增加功能。有了这些新元素，就可以在网页中添加下拉建议项、进度条、工具栏等。这些新元素的问题在于，旧版本的浏览器肯定不支持它们，而鉴于HTML5规范本身还在制定之中，新浏览器也没有急着支持它们。因此，本章只介绍那些已经得到支持的功能。不少读者好奇能用这些元素干什么，但是却不能现在就把它们派上用场，除非你对对付浏览器怪癖和不兼容性上瘾。

4.6.1 使用<datalist>显示输入建议

新的<datalist>元素可以让你在普通文本框中添加一个下拉建议列表。这样，填表的人既可以直接从列表中选择输入，也可以自由输入（参见图4-13）。

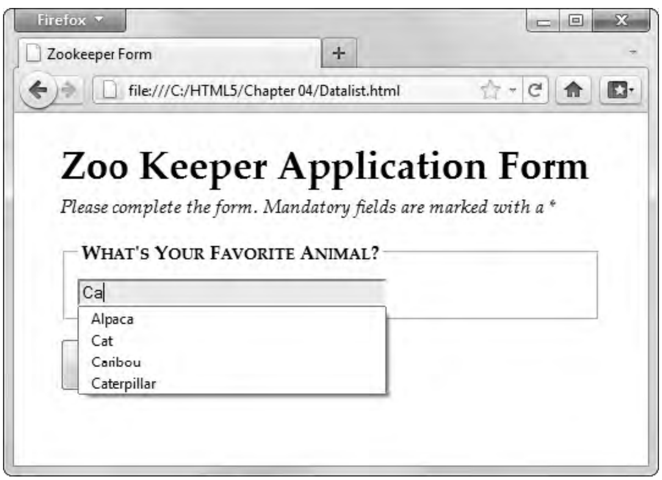


图4-13：输入的同时，浏览器会显示出匹配的建议项。例如，输入字母“ca”，浏览器就会显示名字中包含这两个字母（不一定在开始位置）的动物

<datalist>必须配合一个标准的文本框使用。假设我们有以下<input>元素：

```
<legend>What's Your Favorite Animal?</legend>
<input id="favoriteAnimal">
```

要为这个文本框添加建议项列表，必须先创建一个<datalist>。从技术角度讲，可以在任何地方定义这个列表，因为<datalist>不会显示出来，而只会为使用它的文本框提供数据。话虽这么说，还是把这个<datalist>放在使用它的<input>元素之后（或之前）更合适。下面就是一个<datalist>元素的示例：

```
<datalist id="animalChoices">
  <option label="Alpaca" value="alpaca">
  <option label="Zebra" value="zebra">
  <option label="Cat" value="cat">
  <option label="Caribou" value="caribou">
  <option label="Caterpillar" value="caterpillar">
  <option label="Anaconda" value="anaconda">
  <option label="Human" value="human">
  <option label="Elephant" value="elephant">
  <option label="Wildebeest" value="wildebeest">
  <option label="Pigeon" value="pigeon">
  <option label="Crab" value="crab">
</datalist>
```

与原来的<select>元素一样，<datalist>也使用<option>定义数据项。每个<option>表示一个可供选择的建议，其label属性是显示在文本框中的内容，而value属性是最终会发送给服务器的值（如果用户选择了该项）。就其本身而言，<datalist>是完全不可见的。为了将它与文本框联系起来以便提供建议，下一步就需要将<input>元素的list属性设定为<datalist>的ID：

```
<input id="favoriteAnimal" list="animalChoices">
```

当前版本的Chrome、IE、Firefox和Oprea支持<datalist>。它们会显示如图4-13所示的可能匹配列表。但是Safari、旧版的IE（IE9及其更早版本）以及移动浏览器会忽略list属性和<datalist>元素，所有建议项也就白定义了。

但也不尽然，我告诉大家一个不错的后备技巧，可以让其他浏览器也能利用这些数据。技巧就是在<datalist>中再添加另一个元素。这个技巧之所以可行，是因为支持<datalist>的浏览器只会关注其中的<option>元素，而会忽略其他内容。下面这个修改后的例子就利用了这一点。（支持<datalist>的浏览器会忽略其中的粗体标记。）

```
<legend>What's Your Favorite Animal?</legend>
<datalist id="animalChoices">
  <span class="Label">Pick an option:</span>
  <select id="favoriteAnimalPreset">
   <option label="Alpaca" value="alpaca">
    <option label="Zebra" value="zebra">
    <option label="Cat" value="cat">
    <option label="Caribou" value="caribou">
    <option label="Caterpillar" value="caterpillar">
    <option label="Anaconda" value="anaconda">
```

```
      <option label="Human" value="human">
      <option label="Elephant" value="elephant">
      <option label="Wildebeest" value="wildebeest">
      <option label="Pigeon" value="pigeon">
      <option label="Crab" value="crab">
    </select>
    <br>
    <span class="Label">Or type it in:</span>
  </datalist>
  <input list="animalChoices" name="list">
```

删除上面的粗体标记，结果与前面的例子完全一样。而这样一来，支持<datalist>的浏览器仍然只会显示一个文本框和一个下拉建议项列表（与图4-13显示的一样）。而在其他浏览器中，新添加的标记会把<datalist>的那些建议项组织成一个选择列表，并允许用户选择、输入（如图4-14所示）。

这种过渡完全没有痕迹。只不过在服务器端接收到表单数据后，需要判断数据是来自选择列表（即这里的favoriteAnimalPreset）还是来自文本框（即favoriteAnimal）。虽然会多费这一点点周折，但毕竟为用户提供了很大的方便，没有抛弃任何人。

注意 最初引入<datalist>元素的时候，还为它设计了一个从其他地方（如Web服务器，然后可能再访问数据库）取得数据的功能。在HTML标准未来的版本中，可能还会正式增加这项功能。不过现在要想实现这项功能，恐怕还只能自己写JavaScript代码，利用XMLHttpRequest对象（参见11.1.1节）来获得数据。



图4-14：在不支持<datalist>的浏览器中也可以使用建议项，但需要把建议项封装在一个<select>列表中

4.6.2 进度条和计量条

另外两个新图形微件是<progress>和<meter>，这两个元素外观相似，作用不同（参见图4-15）。

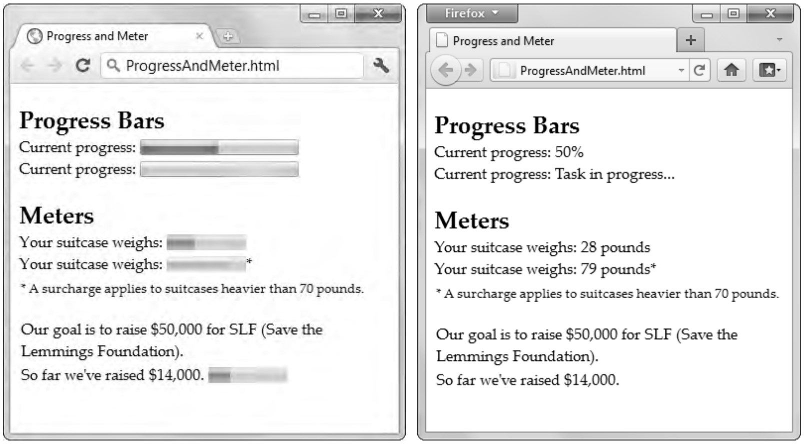


图4-15：在支持的浏览器中，<meter>与<progress>能够提供形象的度量（左）。而在其他浏览器中，则会显示你设定的后备内容（右）

其中，<progress>表示任务的进度，背景为灰色，完成的部分填充为脉动式绿色条。说起这个形象，大家可能都很熟悉，Windows操作系统中复制文件时就会出现这种进度条。不过，用户查看网页的浏览器不同，进度条的样子也可能不同。

而<meter>元素表示的是位于已知范围内的一个值。乍一看，<meter>与<progress>的外观相同，但实际上绿色条阴影更深一些，而且没有脉动效果。根据浏览器不同，计量条的颜色可能会在值“过低”或“过高”的时候改变。比如，图中后一种情况下，Chrome把绿条变成了黄条。但<meter>与<progress>最大的不同，还是标记要表达的语义。

注意 严格来讲，新的<meter>与<progress>元素并非必须出现在表单中。实际上，它们甚至都不是真正的控件（因为它们不能从网页访客那里收集信息）。可是，官方的HTML5规范把它们归为一类，也是考虑<meter>与<progress>元素给人的感觉像是控件（以图形的方式显示数据）。

所有主流浏览器的最新版本支持<meter>与<progress>元素。但是，在旧版的IE（IE9及其更早版本）和一些移动浏览器上会有麻烦。为确保对所有浏览器都支持，需要用一些像HTML5Forms（4.4.2节）之类的东西来兼容这个功能。

使用<meter>与<progress>很简单。先来看看<progress>，它有一个value属性，用于设置表示进度的百分比（即填充的绿色条的宽度），值其实是0到1之间的小数。比如，可以用0.25来表示完成了进度的25%：

```
<progress value="0.25"></progress>
```

另外，也可以利用max属性设置最大值，改变进度条的比例。例如，max设为200，那么value就要位于0和200之间。如果把value设为50，那么结果就和前面例子中将value设置为0.25（25%）一样：

```
<progress value="50" max="200"></progress>
```

这个比例其实就是为了让人看着方便。而浏览网页的人也不会看到进度条中实际的值。

注意 实际上，<progress>元素只是一种显示立体进度条的便捷方式，这个元素本身什么也不会做。比如，要是想利用进度条来显示后台任务（比如，使用13.2节中的Web Worker）的进度，那么你必须自己编写JavaScript代码取得<progress>元素并实时更新它的值。

不支持<progress>元素的浏览器会忽略它。作为后备，可以在这个元素内部放置进度值，如：

```
<progress value="0.25">25%</progress>
```

记住，在支持<progress>元素的浏览器中，是不会显示这个后备内容的。

除了实际显示进度的进度条，还有另外一种进度条——不确定进度条。不确定进度条表示反正有后台任务，但到底什么时候完成不知道（可以想象那些永远也不停转的GIF图）。不确定进度条也是灰色背景，但不断会有绿色闪过，从左到右。要创建不确定进度条，只要不设置value值即可：

```
<progress>Task in progress ...</progress>
```

<meter>元素大致也一样，只不过它表示的是某种计量，因此也被称为计量器。一般来说，给<meter>元素设置的值都会对应现实中的某个值（比如，钱数、天数或重量）。为了控制<meter>元素显示这些数据的方式，需要设置一个最大值和一个最小值（使用max和min属性）：

```
Your suitcase weighs: <meter min="5" max="70" value="28">28 pounds</meter>
```

同样，位于<meter>元素开始与结束标记之间的内容，只会在不支持该元素的浏览器中显示。当然，有时候把<meter>元素的值显示出来也是必要的。此时，你得自己把这个值添加到页面中，不要依赖提供后备内容的方式。下面的代码展示了相应的做法，即先显示出所有信息，然后再添加一个可选的<meter>元素（只有支持它的浏览器才会显示）：

```
<p>Our goal is to raise $50,000 for SLF (Save the Lemmings Foundation).</p>
<p>So far we've raised $14,000. <meter max="50000" value="14000"></meter>
```

为了让<meter>元素能够表示那些“过高”或“过低”的值，而且还能表示得恰如其分，就需要用到low和high属性。大于high（但小于max）的值，就说明超过了它应有的大小了，但仍然是可以接受的。类似地，小于low（但大于min）的值就是过低了：

```
Your suitcase weighs:
<meter min="5" max="100" high="70" value="79">79 pounds</meter>*
<p><small>* A surcharge applies to suitcases heavier than 70 pounds.
</small></p>
```

有些浏览器可能不会利用这些信息。比如，Chrome对于过高的值会显示黄条（参见图4-15），

但对于过低的值则没有任何变化。最后，还可以使用optimum属性将某个值标记为理想的值，但这个属性不会影响计量器在当前浏览器中的显示结果。

总之，只要浏览器支持，<progress>和<meter>就可以为用户带来一些便利。

4.6.3 使用<command>和<menu>创建工具条和菜单

这个功能也许是所有未实现功能中最有用的。设计思路就是通过一个元素（<command>）来表示用户可以执行的操作，而用另一个元素（<menu>）来封装这组操作。灵活组织这两个元素并为它们设置适当的样式，可以利用<menu>把Mac桌面下方的可停靠工具条搬到浏览器窗口中来，或者创建出单击后显示的弹出式上下文菜单。可是，现在还没有浏览器支持这两个元素，所以要知道它们会不会带来我们想象的效果，只能拭目以待。

4

4.7 网页中的 HTML 编辑器

第1章我们就讨论过，HTML5奉行“修补牛蹄子路”的原则。意思就是说，把今天开发人员使用的未标准化的功能，正式写入HTML5标准。这方面的一个例子就是标准化了两个奇怪的属性：contenteditable和designMode。这两个属性的作用是将浏览器转换成简单的HTML编辑器。

这两个属性早就已经有了。事实上，它们是在原先IE一统天下的时候，由IE5率先引入的。随着越来越多Windows扩展的出现，大多开发人员都不再使用这两个属性了。但随着时间推移，其他浏览器也陆续支持了IE实用但又怪异的富HTML编辑功能。今天，所有桌面浏览器都支持这两个从未写进任何标准的属性。

何时使用HTML编辑功能

在尝试富HTML编辑功能之前，有必要先搞清楚这个功能到底有什么用。除了能带来新奇感之外，能编辑HTML实际上对任何人都没有什么吸引力。除非你需要向用户提供一种简单快捷的编辑HTML内容的方式，比如让用户能添加博客文章、输入评论、发布分类广告或者编写发送给其他用户的消息。

即使你确定需要这种功能，contenteditable和designMode属性也未必是第一选择。因为它们不能提供真正的网页设计工具所具备的那些好用的功能，比如修改标记、查看和编辑HTML源代码、拼写检查，等等。使用HTML的编辑功能，确实可以构建更好用的编辑器，但需要做一些额外的工作。可是，如果你真需要富文本编辑功能，恐怕还是选择别人已经做好的编辑器更方便，只要把相应代码插入网页中即可。流行的富文本编辑器有TinyMCE（www.tinymce.com）和CKEditor（http://ckeditor.com）。

4.7.1 使用contenteditable编辑元素

下面要介绍的第一个能帮我们实现HTML编辑功能的属性是contenteditable。把这个属性添

加到任何元素并且设置它的值为true以使该元素的内容可以编辑：

```
<div id="editableElement" contenteditable="true">You can edit this text, if
you'd like.</div>
```

乍一看，可能还看不出有什么不同。但加载完网页后再单击这个<div>元素的内容，你就会发现文本编辑光标（插入光标），如图4-16所示。

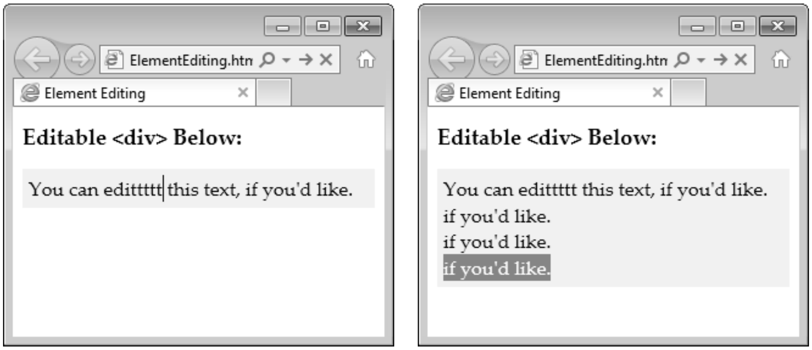


图4-16：单击可编辑区域后，就可以通过键盘上的前后左右键移动光标，可以删除文本，也可以插入内容（左）。还可以按Shift键选择并复制、剪切、粘贴文本（右）。很像在Word中编辑文本，只是不能超过<div>的界限

在这个例子中，可编辑的<div>中只包含文本，但其实可以把任何元素放入其中。就算是把整个页面放到里面，让用户可以编辑整个页面也没有问题。类似地，要想让页面中几个不同部分可以编辑，只要为相应元素应用contenteditable属性即可。

提示 有些浏览器支持少量的内置命令。比如，在IE中使用快捷键Ctrl+B、Ctrl+I和Ctrl+U可以为文本加粗、加斜体和加下划线。类似地，在Firefox中，按Ctrl+Z可以撤销上一次操作。而在Chrome中可以使用前述所有命令。要想进一步了解这些编辑命令，以及如何创建可以触发它们的自定义工具条，请参考Opera的两篇文章：http://tinyurl.com/htmlEdit1和http://tinyurl.com/htmlEdit2。

通常，我们都不会在标记中设置contenteditable属性，要设置也是通过JavaScript，并且在编辑完成后再取消可编辑的功能。下面这两个函数就是用来开启和关闭编辑功能的。

```
function startEdit() {
  //让元素可以编辑
  var element = document.getElementById("editableElement");
  element.contentEditable = true;
}

function stopEdit() {
  //把元素修改为正常状态
  var element = document.getElementById("editableElement");
  element.contentEditable = false;
```

```
  //在消息框中显示标记
  alert("Your edited content: " + element.innerHTML);
}
```

以下两个按钮用于触发它们：

```
<button onclick="startEdit()">Start Editing</button>
<button onclick="stopEdit()">Stop Editing</button>
```

不要把这两个按钮放到网页的可编辑区域中！因为网页一变得可以编辑，其中的元素就不会产生事件，因而就无法再触发JavaScript代码了。

图4-17展示了元素变成可编辑之后和为其中内容应用一些样式后的结果（拜Ctrl+B命令所赐）。

注意 不同浏览器中的富HTML编辑功能也会有一点差异。例如，在Chrome中按Ctrl+B会为元素添加<b>标签，而在IE中则会添加<strong>标签。而在按回车键换行和按退格键删除标签时，也会出现差异。说到这，就不难理解HTML5标准化富HTML功能的的意义了，至少可以让不同浏览器的行为一致。



图4-17：这个图证明了编辑元素的确可以改变页面在内存中的信息。在这个例子中，新内容显示在了弹出框里。但在实际应用中，这些数据会被发送给Web服务器，或许还会用到11.1.1节介绍的XMLHttpRequest对象

4.7.2 使用designMode编辑页面

与contenteditable属性类似，但designMode属性能够让用户编辑整个页面。你也许会问：如果让整个页面都可以编辑，那么用户还怎么单击按钮，我们还怎么控制编辑过程呢？当然有办法，那就是把要编辑的文档放在一个<iframe>元素中，而这个元素就充当了一个超级的编辑框（见图4-18）。



图4-18：这个页面中包含两个框，第一个是<iframe>，其中显示着第2章的启示录页面。第二个是普通的<div>，其中显示编辑后的启示录页面的HTML标记。页面中的两个按钮控制着显示，用于将<iframe>切换为设计模式

这个页面的标记十分简单。以下就是这个页面<body>元素中的所有内容：

```
<h1>Editable Page</h1>
<iframe id="pageEditor" src="ApocalypsePage_Revised.html"></iframe>
<div>
  <button onclick="startEdit()">Start Editing</button>
  <button onclick="stopEdit()">Stop Editing</button>
</div>

<h1>Edited HTML</h1>
<div id="editedHTML"></div>
```

显然，这个例子有赖于startEdit()和stopEdit()方法，这两个方法与前面的示例类似。只不过这里的代码修改的是designMode属性，而非contenteditable属性：

```
function startEdit() {
  //把<iframe>转换为设计模式
  var editor = document.getElementById("pageEditor");
  editor.contentWindow.document.designMode = "on";
}

function stopEdit() {
  //关闭<iframe>的设计模式
  var editor = document.getElementById("pageEditor");
  editor.contentWindow.document.designMode = "off";
```

```
  //显示编码后的HTML（仅为验证确实可以编辑）
  var htmlDisplay = document.getElementById("editedHTML");
  htmlDisplay.textContent = editor.contentWindow.document.body.innerHTML;
}
```

通过这个例子可以更好地体验富文本编辑功能。例如，单击图片就可以在浏览器中操作它。可以调整它的大小，把它拖到新位置，或者单击之后按删除键就可以把它删除。如果页面中有表单控件，那么对它们也可以执行相同的操作。

当然，要想把这个例子变得更实用还需要做很多工作。第一，需要添加更直观的编辑控件。在此，我还要推荐Opera乐于助人的开发人员写的那两篇文章（http://tinyurl.com/htmlEdit1和http://tinyurl.com/htmlEdit2），文章深入解释了命令模型，但超出了本章的范围。第二，需要妥善处理编辑后的标记，比如通过XMLHttpRequest（参见12.1.1节）把它们发送给Web服务器。

最后还要提醒读者，如果你是在本地硬盘上运行上面的例子，在有些浏览器中可能会遇到问题。（IE和Chrome会启用安全限制，而Firefox则一帆风顺，不会有任何问题。）为了避免出现问题，可以在http://prosetech.com/html5/上运行这个例子。

4

Part 2 第二部分

视频、图形和特效

本部分内容

第 5 章

音频与视频

最早的时候，互联网主要用于分享学术研究的成果。情况很快就发生了改变，Web似乎一夜之间就成了万众瞩目的焦点，成了新闻业和商业发展背后的动力之源。今天，互联网最先进的网络技术较少用于物理计算，而更多地用于在全球范围内传播猫咪演奏钢琴的病毒视频。网络巨头思科的报告指出，这个趋势没有放慢的迹象，并预测到2017年网络流量的80%都将用于视频。

令人难以置信的是，这种巨大的转变居然是在HTML语言不内置支持视频，甚至连音频都不支持的现实下发生的。在不久的过去，网民依赖于Flash，这个插件大多数时候对大多数人都是有效的。但是Flash有一些关键的短板，其中包括苹果设备（比如iPhone和iPad）不支持。

为了解决这些问题，HTML5添加了`<audio>`和`<video>`这两个HTML多年来一直缺少的元素。然而，向HTML5音频和视频的过渡远没有那么顺利。数年来浏览器厂商身陷关于格式的口诛笔伐的战争。好消息是，今天大部分战争已尘埃落定，并且HTML5音频和视频对即便非常谨慎的开发者也是很好的选择。

5.1 网络视频的演变

在没有HTML5的情况下，可以通过两种方式向网页中添加视频。一种过时的方式是使用`<embed>`元素把视频硬塞进页面中。然后，浏览器就可以使用Windows Media Player、Apple QuickTime或其他视频播放器创建一个视频窗口，并把它放在页面中。

这种方式的问题是一切只能听天由命。你没有办法控制播放进度，也不能提前缓冲视频以避免长时间的播放停滞，甚至你都不知道自己的视频文件能否在不同浏览器或操作系统中播放。

第二种方式是使用浏览器插件，比如微软最近推出的Silverlight或最受欢迎的Adobe Flash。Flash完全解决了浏览器支持问题，Flash视频能够在安装了Flash插件的任何地方播放；用数字来说，就是目前能上网的计算机中有99%都安装了Flash播放器。Flash为我们提供了几乎无限制的控制功能，而且我们还可以方便地使用别人做好的Flash播放器，甚至每个发光按钮你都可以自己重新设计。

不过，Flash也不完美。为了把Flash视频放到网页中，必须使用`<object>`和`<embed>`元素编写一大堆乱七八糟的标记，必须适当地编码视频文件，可能还必须要花高价购买Flash开发软件并学

习使用——轻易学不会。但是，最严重的问题还在于苹果的移动设备iPhone和iPad。它们根本就不支持Flash，因而通过它们查看内置Flash视频的页面，只能看到一个空白的方框。

注意 插件不可靠也是业界的共识。这也是插件的工作方式决定的。比如，在访问使用Flash的页面时，浏览器会把页面中的某个矩形区域交给Flash控制。多数情况下，交接工作的过程都比较顺利。可是，一些小bug的存在，或者异常的系统配置，都可能导致意外的通信和故障，因而造成视频混乱或消耗大量计算机内存，最终让上网变得缓慢无比。

尽管如此，今天上网，不用iPhone或iPad，你所看到的视频，仍然都被包装在迷你Flash应用中。你不信？右键单击视频播放器看一看。如果上下文菜单中包含“About Flash Player 11”字样的命令，那恭喜你，你点的就是无所不在的Flash插件。而且，即使是使用HTML5，恐怕我们还要准备一个Flash视频作为后备文件，以应付那些落后的浏览器（比如IE8）。

注意 YouTube提供一个试用版HTML5视频播放器。要想体验，请访问www.youtube.com/html5。但在其他地方，YouTube仍然只使用Flash。除非你在用iPhone或iPad，在这种情况下，YouTube会智能地自动切换到支持HTML5视频。

5.2 HTML5 音频与视频

HTML5支持音频和视频的想法非常简单。既然能使用`<img>`元素在网页中添加图像，就应该能使用`<audio>`元素和`<video>`元素在网页中添加音频和视频。是这个道理，于是HTML5就增加了这两个元素。

不行就还用Flash

HTML5 新增的音频和视频功能不能满足所有需求。如果你需要考虑以下事项，那么最好还是用 Flash（至少目前还是要用）。

- **有许可限制的内容**。HTML5 视频文件没有任何版权保护措施。事实上，任何人都可像下载图片一样下载 HTML5 视频，只要右键单击即可。那就是说，数字版权管理功能现在还在开发中，并且计划包含在 HTML 5.1 中。
- **录制视频或音频**。HTML5 不支持从一台电脑到另一台电脑传送音频或视频流。如果你想开发一个在线聊天程序，要使用访客机器上的麦克风和摄像头，还得用 Flash。HTML5 制定者为实现相同功能，正尝试新增`<device>`元素。但目前在任何浏览器中，都没有办法只通过 HTML 实现此功能。
- **自适应视频流**。主流的、视频丰富的网站，比如 YouTube，都需要精细地控制视频流和缓冲。这些网站需要以不同解析度提供视频、进行实况直播、根据访客的带宽调整视频

质量。在 HTML5 能提供这些功能之后，视频分享网站可能会向 HTML5 迁移，但不会完全脱离 Flash。

- **低延迟、高性能音频**。有些应用需要音频一开始就不能间断，或者需要同时播放的多个音频之间完美配合。比如虚拟合成器、音乐观察器或者多种音效同时播放的实时游戏。虽然浏览器开发商在努力提升 HTML5 音频的性能，但目前还无法满足要求。
- **动态创建或编辑音频**。如果你不只是想要播放录制好的音频，而是还需要分析、修改音频信息，然后实时生成音频，怎么办？确实有一些新的标准，比如处在试验阶段的 Web Audio API（http://tinyurl.com/web-audio-API），可以为 HTML5 补充这部分功能，但目前还是指望不上啊。

5.2.1 使用<audio>播放点噪音

以下是使用<audio>元素的一个最简单的例子：

```
<p>Hear us rock out with our new song,
<cite>Death to Rubber Duckies</cite>:</p>
<audio src="rubberduckies.mp3" controls></audio>
```

这里的src属性是要播放的音频文件的文件名。而controls属性告诉浏览器要包含基本的播放控件。每个浏览器中的播放控件都不太一样，但用途都一样，都可以控制开始和结束，跳到新位置和调节音量（图5-1）。

注意 除了基本的src和controls属性，<audio>元素还支持一些其他属性，这会在接下来的几节中详细介绍。

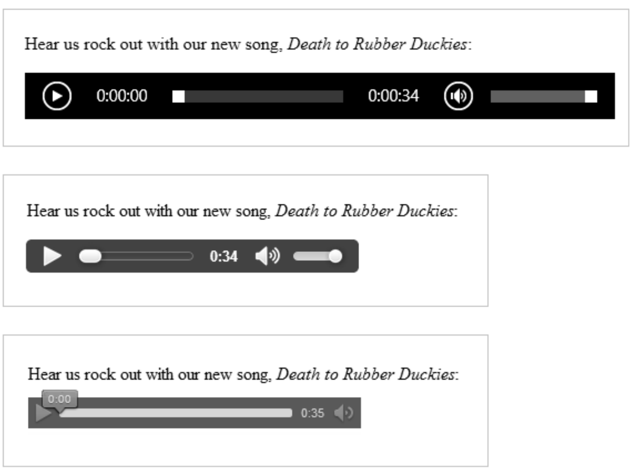


图5-1：这里是IE（上）、Chrome（中）和Firefox（下）中的播放控件

5.2.2 预加载媒体文件

preload是一个有用的属性，它告诉浏览器应该怎样加载一个媒体文件。指定preload的值为auto，让浏览器下载整个文件，以便用户单击播放按钮时就能播放。当然，下载过程是后台进行的，网页访客不必等待下载完成，而且仍然可以随意查看网页。

除了auto之外，preload属性还支持另外两个值：metadata和none。前者告诉浏览器先获取音频文件开头的数据块，从而足以确定一些基本信息（比如音频的总时长）。后者告诉浏览器不必预先下载。恰当地利用这些值，可以节省带宽。比如，当页面中有很多<audio>元素，而你又不认为访客会播放其中很多音频的时候，就可以有选择地使用前述三个值。

```
<audio src="rubberduckies.mp3" controls preload="metadata"></audio>
```

如果使用的是none或metadata，那么浏览器会在用户单击播放按钮时立即下载音频文件。通常，浏览器在下载后续数据时，会播放已经下载完的部分；除非你的网速很慢，否则应该不会卡。

5

如果没有设置preload属性，浏览器就自己决定是否预先下载了。对这一点，不同浏览器的处理方式也不一样。多数浏览器将auto作为默认值，但Firefox的默认值是metadata。不过，也请大家注意，这个preload属性也不是必须严格执行的规则，而只是你对浏览器的建议。根据具体情况，浏览器可以忽略你的设置。（有些旧版本浏览器根据不会在意preload属性。）

注意 如果页面中有很多<audio>元素，浏览器会分别为它们创建自己的播放控件。访客可以每次只播放一个音频文件，也可以同时播放多个。

5.2.3 自动播放

接下来再看看autoplay属性。这个属性告诉浏览器在加载完音频文件后立即播放：

```
<audio src="rubberduckies.mp3" controls autoplay></audio>
```

如果不设置autoplay属性，必须是用户单击播放按钮才会播放音频文件。

可以利用<audio>元素不知不觉地播放背景音乐，或者为浏览器游戏播放音效。要实现背景播放，去掉controls属性，加上autoplay属性就好了（或者利用JavaScript来控制播放，参见5.4.1节）。不过要注意，即使开启背景播放，也要在页面中提供相应的装置，以便用户能够关闭声音。

注意 谁也不愿意浏览一个播放难听的背景音乐，但却无法关闭其声音的网页。如果你没有给<audio>元素添加controls属性，那必须或至少要添加一个静音按钮，利用JavaScript让用户能够设置静音。

5.2.4 循环播放

最后，loop属性告诉浏览器在播放结束时，再从头开始重新播放：

```
<audio src="rubberduckies.mp3" controls loop></audio>
```

大多数浏览器都可以流畅地循环播放音频文件，因此可以利用这一点创建没有穷尽的音乐播放体验。关键在于选择一段终点与起点恰好能够衔接起来的音频片段。类似这样的片段，访问http://www.flashkit.com/loops/可以找到很多。（这些可循环文件是为Flash设计的，但也可以下载到MP3和WAV格式的。）

要是你觉得<audio>元素实在是太好了，嗯，我也承认。5.3节将介绍让HTML5开发人员头痛欲裂的格式问题。不过，在头疼之前，我还得先给你介绍<audio>元素的亲密战友：<video>元素。

5.2.5 了解<video>

<video>与<audio>实现太相像了。它们有相同的src、controls、preload、autoplay和loop属性。下面就是一个直观的例子：

```
<p>A butterfly from my vacation in Switzerland!</p>
<video src="butterfly.mp4" controls></video>
```

同样，controls属性告诉浏览器生成方便的播放控件（见图5-2）。在大多数浏览器中，单击页面其他任何地方，播放控件都会自动隐藏，而当鼠标悬停于影片画面上时，它们又会显示出来。

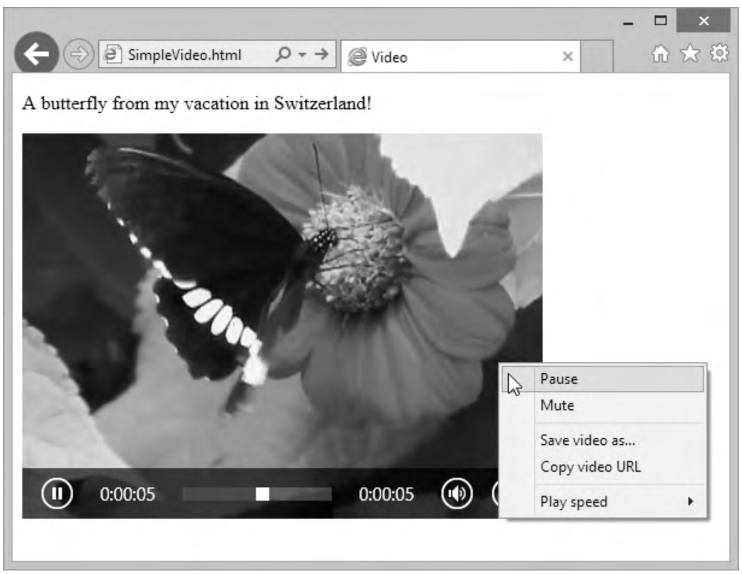


图5-2：很容易把<video>元素当成Flash视频窗口。但在<video>元素上右击鼠标，会看到比Flash更简单的菜单，其中包含把视频文件保存到本地的命令。在有的浏览器里，菜单中可能还会包含改变播放速度、循环播放视频、全屏播放及静音等选项。

<video>元素和<audio>元素一样，也有src、controls、preload、autoplay和loop属性。可是，

如果启用了自动播放，要使它不那么招人讨厌，可以将其设置为muted状态，这在大部分浏览器上会关闭声音。通常，用户可以点击扬声器图标来重新打开声音。

除了与<audio>共同的属性之外，<video>元素还有另外3个属性：height、width和poster。

其中，height和width属性用于设置视频窗口的（像素）大小。下面这行代码会创建一个400像素×300像素的视频窗口：

```
<video src="butterfly.mp4" controls width="400" height="300"></video>
```

在设置这个尺寸时，应该注意按照视频的原始比例设置。而明确设置视频窗口大小，可以在视频尚未加载完成时（或者视频加载失败时），不影响页面的布局。

注意 不论怎样设置视频容器的尺寸，视频画面总是会保持自身的比例。比如，如果将一个400×300像素的视频放在一个800×450像素的视频容器中，将获得一个未拉伸且适应容器的最大视频画面，即600×450像素。这会在视频画面的两边各留出100像素的空白。

5

最后，poster属性用于设置替换视频的图片。浏览器在三种情况下会使用这个图片：(1) 视频第一帧未加载完毕；(2) 把preload属性设置为none；(3) 没有找到指定的视频文件。

```
<video src="butterfly.mp4" controls poster="swiss_alps.jpg"></video>
```

好了，我们现在已经介绍完有关HTML5音频和视频标记的所有内容了。然而，通过巧妙地使用JavaScript，实际上还有更多可能性。不过，我们已经不能再回避了，在继续讨论利用<audio>和<video>元素做一些吸引人的东西之前，无论如何得直面令人头疼的音视频编解码问题。

媒体组

HTML5标准指定了一个不常用的属性mediagroup，它适用于<audio>和<video>元素。可以用mediagroup属性将多个媒体文件连接到一起，这样它们的播放就是同步的。只需给每个<audio>和<video>元素赋予相同的mediagroup名称（可以是任何想要的值）：

```
<video src="shot12_cam1.mp4" controls
mediagroup="shot12"></video>
<video src="shot12_cam2.mp4" controls
mediagroup="shot12"></video>
```

现在，如果用户在第一个视频窗口点击播放（shot12_cam1.mp4），两个窗口都会立刻开始播放。

mediagroup属性可以用于同步同时发生的视频文件，比如，从不同角度拍摄的体育赛事录像。也可以用它来同步音频和视频，这对需要基于用户的语言或无障碍访问需求来选择不同的音轨很有用。比如，可以给针对视觉障碍者的音轨添加一个描述情节进展的画外音。为此，需要在页面中隐藏多个<audio>元素，每个都取一个不同的mediagroup名称，然后添加一些简单的JavaScript，用以基于用户的需求设置<video>元素的mediagroup名称来匹配相应的<audio>元素。

不幸的是，mediagroup现在还不是很有用，因为支持它的浏览器有限。Chrome和Opera支持，但最新的IE和Firefox完全不支持。

5.3 HTML5 媒体格式

<video>和<audio>元素是不是太好用了？是的，有时候它们是这样的。问题是在一个浏览器上运行得很好的媒体格式可能在另一个浏览器上运行得很糟。

我们刚刚列举的示例使用了两种流行的标准：MP3音频和H.264视频。这两种格式是大部分浏览器的最爱，但在Opera浏览器上，它们不能运行（图5-3）。

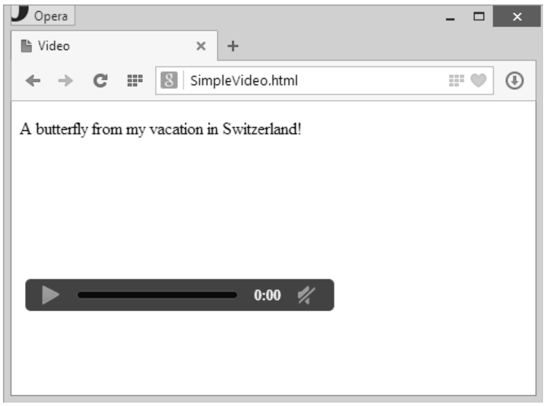


图5-3：如果在Opera中加载一个使用了H.264视频的页面，播放控件是不可用的，在本该出现视频的地方会出现空白

幸运的是，可以用一个格式后备措施来解决这个问题，可参见5.4.1节。但在学习如何做之前，需要更深入地了解一下今天Web中音频和视频格式的范围，以及目前浏览器的支持状况。

5.3.1 谈谈格式

官方HTML5标准没有要求浏览器支持任何一种视频或音频格式。（之前的版本要求过，但最后经过激烈的讨论后还是删除了。）因此，浏览器开发商可以自由选择想要支持的格式，而事实上他们骨子里就不可能达成一致。表5-1展示了目前不同浏览器使用的标准。

表5-1　某些HTML5浏览器支持的音频和视频标准

格　　式	说　　明	常用扩展名	MIME类型
MP3	世界上最流行的音频格式	.mp3	audio/mp3
Ogg Vorbis	免费、开放的标准，能够提供高质量的压缩音频，可以与MP3媲美	.ogg	audio/ogg
WAV	未加工数字音频的初始格式。由于未经压缩，所以体积奇大，大多数情况下不适合Web	.wav	audio/wav
H.264	视频压缩的行业标准，特别适合高清晰度视频。广泛应用于消费设备（如蓝光播放器和便携式摄像机）、Web分享站点（如YouTube和Vimeo）和Web插件（如Flash和Siverlight）	.mp4	video/mp4
Ogg Theora	免费、开放的视频标准，出自Vorbis音频标准的制定者之手。品质和性能不及H.264，但可以满足大多数人的需要	.ogv	video/ogg

（续）

格　式	说　明	常用扩展名	MIME类型
WebM	最新的视频格式，谷歌在买下VP8之后，将其改为免费标准。有评论指出，其品质尚不如H.264，而且可能牵涉其他人的专利，因此将来或许会引发诉讼。	.webm	video/webm

表5-1中也列出了媒体文件应有的扩展名。为什么扩展名很重要？回答这个问题，必须认识到一个视频文件实际上有三个标准参与其中。首先，也是最明显的，就是视频编解码器，用于把视频压缩为数据流（包括H.264、Theora和WebM）。其次是音频编解码器，利用相关的标准压缩一或多个音频文件。（例如，H.264一般使用MP3，而Theora则使用Vorbis。）第三是容器的格式，规定了如何把视频、音频、描述性信息以及静态图片和字幕等填充物组合到一起。一般来说，文件的扩展名表示容器的格式。因此，.mp4意味着MPEG-4容器，.ogv表示Ogg容器。

比较微妙的地方在这里：多数容器格式都支持一些不同的视频和音频标准。例如，流行的Matroska容器（.mkv）可以包含H.264或Theora编码的视频。考虑到不让你的脑袋疼得裂开，表5-1针对每种视频格式给出了一种最常用，同时也是在Web上获得了最可靠支持的容器格式。

表5-1中也列出适当的MIME类型，这些在配置Web服务器的时候有用。如果不小心用错了MIME类型，那么浏览器可能就会顽固地拒绝播放本身好好的媒体文件。（要是你还不太明白什么是MIME类型，如何配置，可以参考下一节的内容。）

MIME类型及其使用原因

MIME类型（有时候也叫**内容类型**）就是一小段信息，表示某种Web资源的内容类型。例如，网页的MIME类型就是text/html。

Web服务器在把某个资源发送给浏览器的时候，会在前面发送MIME类型。比如，假设浏览器请求了一个网页SuperVideoPlayerPage.html，服务器会发送text/html这个MIME类型、其他一些相关信息和实际的文件内容。浏览器接收到该MIME类型后，就知道该如何处理后面的内容。这样，就不必根据文件的扩展名或其他信息去判断了。

对于常见的文件类型（如HTML页面和图片），不必担心其MIME类型，因为Web服务器都能够适当地处理。但某些Web服务器或许没有配置音频和视频的MIME类型。这就是问题了，如果Web服务器在发送媒体文件时发错了MIME类型，会导致浏览器不知所措。而结果一般就是不能播放。

为避免这个问题，务必要按照表5-1列出的MIME类型正确地配置Web服务器，同时也不要忘了给自己的音频和视频文件使用正确的扩展名。（光正确配置MIME类型还不够，还得使用正确的扩展名。因为Web服务器要成对儿使用这两方面信息。比如，把.mp4文件配置为video/mp4这个MIME类型后，却在视频文件上使用了.mpFour作为扩展名，那Web服务器就不知道你想让它做什么了。）

配置MIME类型并不难，但实际的步骤却因Web托管公司（或者如果你自己管理自己的服务器，就是你的Web服务器软件）而异。如果网站托管公司使用流行的cPanel工具，可以找到

名为MIME Types的图标，单击之后就可以看到如图5-4所示的页面。如果你还有什么疑问，请联系托管公司寻求帮助。

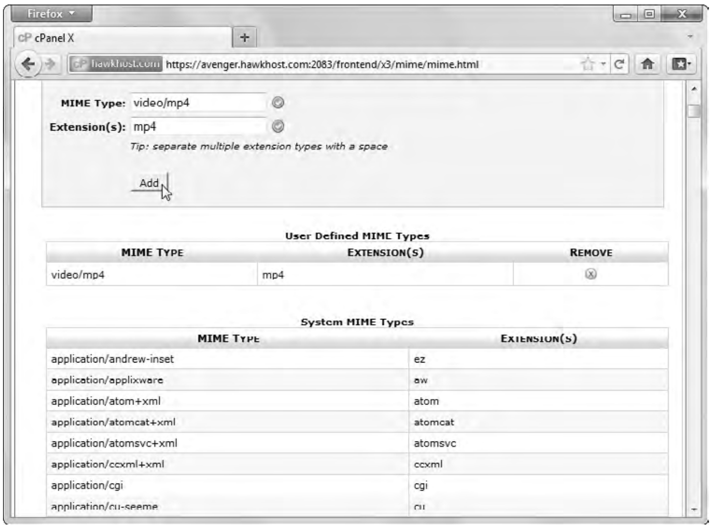


图5-4：在这里我们添加了一个新的MIME类型，以支持H.264视频文件。如果你的网站已经配置好了这个MIME类型，那你就不必多此一举了

5.3.2 浏览器对媒体格式的支持情况

HTML5的格式问题由来已久。浏览器厂商各自不同的需求是冲突的导火索。Mozilla（Firefox开发商）和Opera（Opera浏览器开发商）等小公司，不愿意为MP3音频和H.264视频等流行的标准支付许可使用费。可是，这真的很难责怪他们，毕竟他们自己的开发成果都是允许他人自由使用的。

而大一些的公司，比如微软和苹果，又都有合理的理由不使用没有许可限制的标准。他们报怨那些标准并不完善（目前还没有硬件加速），而且应用得也不够广泛（不像H.264已经广泛应用于便携式摄像机、蓝光播放器和其他很多设备）。实际上，最大的问题还在于，那些没有许可限制的标准可能涉及其他人的知识产权。如果情况确实如此，而且微软、苹果这样的大公司又使用了它们，那么打上一场旷日持久的官司就在所难免了。

幸好，形势在好转。2013年，Firefox妥协并同意支持MP3和H.264。谷歌尽管扬言要在Chrome中移除对H.264的支持，却从未这样做并且现在也不太可能这样做。Oprea仍然是这场博弈的顽固派——到目前为止。浏览器对媒体格式支持的详情，请参见表5-2（音频格式）和表5-3（视频格式）。

表5-2 浏览器对HTML5音频格式的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android
MP3	9	21	5	3.1	–	3	2.3
Ogg Vorbis	–	3.6	5	–	10.5	–	–
WAV	–	3.6	8	3.1	10.5	–	–

表5-3 浏览器对HTML5视频格式的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android
H.264 Video	9	21	5	3.1	–	4*	2.3
Ogg Theora	–	3.5	5	–	10.5	–	–
WebM	–	4	6	–	10.6	–	2.3

* iOS 3.x支持视频，但旧版的Safari浏览器中存在一些微小的视频方面的bug。例如，设置`poster`属性（参见5.2.5节）会导致视频无法播放。

移动设备上的浏览器也各有各的问题。有些不支持像自动播放和循环这样的功能，因为这些功能会耗尽电量以及宝贵的带宽。但即便你不打算使用这些功能，移动设备也需要做专门的考虑来确保良好的视频播放性能以及将数据使用量最小化。针对移动设备进行视频编码时，可以降低质量，可能的话，还可以降低分辨率。

提示 这里有一个经验法则，如果你想让视频能在移动设备上播放，那就应该使用H.264 Baseline Profile（而非Hight Profile）编码。对于iPhone和Android手机，视频尺寸不必超过640像素×480像素（如果还要支持黑莓手机，那么不必超过480像素×360像素）。很多编码软件（参见后面的附注栏）都有针对移动视频进行优化的预设选项。

5

H.264许可

我的视频格式是H.264，我需要付许可费吗?

如果你在自己的产品中使用了H.264解码器（比如，你开发了一款浏览器，能播放H.264编码的视频），那当然得付费。而如果你是提供视频的人，那就要看情况了。

首先，听听不用掏钱的情况。如果你使用H.264制作免费视频，不用交一分钱。如果你制作的是商业视频，但实际上并没有销售（比如拍摄商业广告或通过视频访谈推销自己），那也不用花钱。

如果你在自己的网站上销售H.264编码的视频，那可能就需要向MPEG-LA支付许可费了，要么现在，要么将来。关键的问题是有多少用户。如果用户数少于10万，不用交钱。如果用户数达到10万，但少于25万，那么应该交25 000美元/年。对于一家如此规模的视频销售公司来说，这笔钱似乎还不算多，更何况还有很多其他花销比这要多得多，比如购买专业的编码工具。可是，这个数字在2016年修订许可条款后还会有变化。打算靠Web视频挣钱的大公司更愿意使用开放的、没有许可限制的视频标准，比如Theora或WebM。

要全面了解H.264的许可条款，请参考www.mpegla.com/main/programs/AVC/Pages/Intro.aspx。

5.4 后备措施：如何讨好每一款浏览器

在写作本书的时候，以HTML5 `<video>`元素呈现的H.264格式视频占有80%以上的上网用户。这个比例已相当高，但就其自身而言还不够高。要创建每个人都能看见的视频，需要借助后备措施。

Web开发人员用在HTML5视频上的后备措施有两种。第一种是格式后备措施。这个机制是HTML5内置的，允许你将一种格式的媒体文件（比如 MP3文件）替换成另一种格式的文件（比如Ogg Vorbis）。这种后备措施解决了5.3.1节中提到的Opera的问题。可是，这对不支持HTML5媒体功能的旧版浏览器无效，比如IE8。

第二种后备措施是一个技术方案。如果浏览器不支持<video>和<audio>元素，可以用久经考验的Flash播放器替代。

严谨的Web开发人员会将这两种后备方案都用上。时间紧迫（或者懒一点）的Web开发人员有时会省去格式后备措施，以省去对视频文件重新编码的工作。毕竟，Opera浏览器（唯一不支持H.264的桌面浏览器）只占1%的浏览器使用份额，开发者们推测Opera可能最终会被迫支持H.264。另一方面，Flash后备措施更容易实现，因为它使用相同的媒体文件，并且它兼容了更多的浏览器，比如“恐龙时代”的IE8。所以忽略Flash后备措施是在冒险。

接下来的两小节将介绍这两种后备措施。

5.4.1 支持多种格式

<video>和<audio>元素有一个内置的格式后备系统。要使用它，就要从<video>或<audio>元素中删除src属性，然后嵌套一组<source>元素。以下是一个<audio>元素嵌套<source>元素的例子：

```
<audio controls>
  <source src="rubberduckies.mp3" type="audio/mp3">
  <source src="rubberduckies.ogg" type="audio/ogg">
</audio>
```

在此，一个<audio>元素嵌套了两个<source>元素，每个<source>元素都指向一个不同的音频文件。浏览器会选择播放第一个它所支持的文件。Firefox和Opera会播放rubberduckies.ogg，而IE、Safari和Chrome会播放rubberduckies.mp3。不幸的是，需要由你自己将内容编码成想要支持的每个格式，这是一个浪费时间、CPU和磁盘空间的过程。

理论上讲，浏览器可以通过下载部分文件内容来判断它是否支持相应的格式。但更好的做法则是像这里一样使用type属性提供正确的MIME类型信息（参见上一节）。如此一来，浏览器就只会下载它认为自己能够播放的文件了。（要了解正确的MIME类型，请参考表5-1。）

同样的做法也适用于<video>元素。下面这个例子使用了两个相同的视频文件，但一次用H.264编码，一次用WebM编码，以确保支持所有的HTML5浏览器：

```
<video controls width="700" height="400">
  <source src="beach.mp4" type="video/mp4">
  <source src="beach.webm" type="video/webm">
</video>
```

这个例子有一些新的东西。在使用多个视频格式时，应该把H.264编码的文件放在前头。否则，运行iOS 3.x的iPad无法正确播放该文件。（iOS 4已经修复了这个问题，但将H.264格式的文件放在前头总没有坏处。）

注意 浏览器认为它支持某种类型的音频或视频文件，并不意味着它一定能播放该文件。例如，你可能会对自己的文件使用不常见的高比特率，或者在某种常用的容器格式中使用一种生僻的编解码器。要解决这个问题，可以通过type属性提供类型和编解码器信息，但这样可能会导致标记混乱。HTML5对此给出了详细的描述，请参见http://tinyurl.com/media-types。

如果你的想法更大胆，那可以创建一个同时支持桌面和移动设备的视频页面。这样的话，不仅要考虑H.264和WebM视频格式，还要考虑为硬件配置不高、上网速度慢的设备创建一个窄带宽版本。为确保移动设备播放轻量级视频，桌面浏览器播放高品质视频，需要写一些JavaScript，或者使用媒体查询，这会在8.4节介绍。

编码媒体文件

现在，你应该知道自己想使用哪些格式了。但你不一定知道怎么把媒体文件转换成这些格式。别担心，有很多工具可以帮你。有的工具可以一次转换很多文件，有的工具则以产出高品质内容著称（一分钱一分货），而有的工具是在强大的Web服务器上执行转换，不用等待。关键是根据自己的具体需求，找到适合你的编码工具。

以下向大家推荐几种编码工具。

- **音频编辑器**。如果你想编辑WAV文件，并将它们保存为MP3或Vorbis格式，只要找一个简单的音频编辑器即可。我推荐Audacity（http://audacity.sourceforge.net/），这是一个免费的编辑器，有Mac和Windows版本。不过，要转换成MP3，还要另安装LAME MP3编码器（http://lame.buanzo.com.ar/）。另外，Goldwave（http://www.goldwave.com/）也是一个类似的编辑器，虽然不免费，但价格也只是象征性的。
- **Miro Video Converter**。这个自由、开源的程序有Windows和Mac OS X版，可以将任何视频文件转换成WebM、Theora或H.264，而且还针对iPad、iPhone和Android设备预设了不同大小和格式。唯一不足的地方就是，它没有提供高级调校选项，无法控制编码过程。要试试看？访问http://www.mirovideoconverter.com。
- **Firefogg**。这是一个Firefox插件（http://firefogg.org/），可以创建Theora或WebM视频文件。与Miro相比，它提供了更多选项，而且是在你的浏览器中运行（所有工作都在本地完成，不会涉及Web服务器）。
- **HandBrake**。这是一个开源、多平台的软件（http://handbrake.fr/），能把多种格式转换为H.264（及其他几种最新的格式）。
- **Zencoder**。这是一种专业的、能与你的网站集成工作的媒体编码服务。Zencoder（http://zencoder.com/）能从Web服务器上下载视频文件、将它们转换成你需要的所有格式和比特率，为转换后的文件命名并分门别类地放在相应的位置下。比较大的视频网站可以与Zencoder合作，谈一个合适的按月付费价格。

5.4.2 添加Flash后备措施

格式后备系统有一个关键的限制：只能在支持<audio>和<video>元素（几乎今天市面上的每一个浏览器都可以，除了IE8）的浏览器上运行。要让网页在非HTML5浏览器上运行，需要添加Flash后备措施。

要理解Flash后备措施怎样工作的，首先要知道有史以来的所有浏览器在对待不认识的标签时行为都一样：视而不见。比如，假设IE8遇到了陌生的<video>开始标签，它只会“一笑而过”，根本不去检查其src属性。然而，浏览器不会忽略不认识的元素中包含的内容，这可是一个非常重要的差异。换句话说，对于下面的标记：

```
<video controls width="400" height="300">
  <source src="discoParty.mp4" type="video/mp4">
  <source src="discoParty.webm" type="video/webm">
  <p>We like disco dancing.</p>
</video>
```

不支持HTML5的浏览器看到的只有：

```
<p>We like disco dancing.</p>
```

这也就是我们所说的后备内容。利用这种后备方式，可以向旧版本浏览器用户提供相应的说明，而不会让人感到迷惑不解。

注意 支持HTML5音频的浏览器即使不能播放媒体文件，也会忽略后备内容。例如，如果Opera遇到了一个指向H.264格式的文件但未提供Theora格式的<video>元素，视频播放器不会显示任何内容。

知道了怎么添加后备内容后，接下来就要考虑添加什么内容了。后备内容可以是一句话，比如“你的浏览器不支持HTML5视频，请升级浏览器”。但网站访客认为这种方式极其不礼貌，要是让他们看到这句话，你就永远跟他们无缘了。

比较合适的后备内容是另一段可以播放的视频，也就是要使用一个普通的非HTML5页面。比如，可以使用YouTube视频窗口，但必须遵守YouTube的规定（视频长度不能超过15分钟，而且不能包含令人反感和侵犯版权的内容）。准备好内容后，可以先上传到YouTube，上传时选择一种最佳格式，而YouTube会将其重新编码成它支持的格式。还没有在YouTube上传过视频？先看看这里吧：www.youtube.com/my_videos_upload。

另一种可能是使用Flash视频播放器。（如果你想展示音频，就是Flash音频播放器。）各式各样的Flash播放器太多了，其中很多都是免费的，至少非商业用途不收费。而且，大多数Flash视频播放器还都支持HTML5视频中常用的H.264格式。

下面这个例子展示了把流行的Flowplayer Flash（http://flash.flowplayer.org）插入到HTML5 <video>元素中的标记：

```
<video controls width="700" height="400">
  <source src="beach.mp4" type="video/mp4">
```

```
<source src="beach.webm" type="video/webm">

<object id="flowplayer" width="700" height="400"
  data="flowplayer-3.2.16.swf"
  type="application/x-shockwave-flash">
    <param name="movie" value="flowplayer-3.2.16.swf">
    <param name="flashvars" value='config={"clip":"beach.mp4"}'>
  </object>
</video>
```

代码中的粗体部分是浏览器传递给Flowplayer Flash的参数，包含视频文件的名字。请注意，虽然这个例子考虑了三种可能的情况（H.264格式的HTML5视频、WebM格式的HTML5视频和H.264格式的Flash视频），但只需要两个视频文件，减少了编码工作量。图5-5展示了这个例子的结果。



图5-5：同一个视频，以三种方式提供：对IE9（上）、Firefox（中）使用HTML视频，对IE7（下）使用Flash视频

当然，还是有些人连Flash都没装，而且浏览器也不支持HTML5。对这些用户，可以提供另一种形式的后备内容，比如一个指向视频文件的下载链接，点击可以在外部程序中打开。这样的后备内容要放在Flash内容后面，但仍然在<object>元素内：

```
<video controls width="700" height="400">
  <source src="beach.mp4" type="video/mp4">
  <source src="beach.webm" type="video/webm">

  <object id="flowplayer" width="700" height="400"
   data="http://releases.flowplayer.org/swf/flowplayer-3.2.16.swf"
   type="application/x-shockwave-flash">
     <param name="movie" value="beach.mp4">
     <img src="beach_thumbnail.jpg" alt="A lazy day at the beach">
     <p>Your browser does not support HTML5 video or Flash.</p>
     <p>You can download the video in <a href="beach.mp4">MP4 H.264</a>
      or <a href="beach.webm">WebM</a> format.</p>
  </object>
</video>
```

有趣的是，有另一种实现Flash后备措施的方式。到目前为止，你所看到的例子都以Flash作为HTML5的后备措施，即给每个用户提供HTML5视频（或音频），除了那些使用旧浏览器的用户，他们会获得Flash。可是，可以将这个方案反转一下，优先使用Flash，HTML5作为后备措施。这会给所有用户提供Flash，除了那些没有安装Flash的。如果你的网站已经有成熟的Flash播放器来呈现视频内容，但想拓展到iPad和iPhone用户，这种策略就很合适。如果你的媒体要求超出了HTML5当前的支持范围（如5.2节附注中所介绍的），也可以选择这种方案。

假如你想主要提供Flash视频，以HTML5作为后备那在前面的例子上简单调整一下就好。首先是<object>元素，然后在其中嵌套<video>元素，最后是</object>标签结束。这种情况下，需要把后备内容放在最后一个<source>元素之后：

```
<object id="flowplayer" width="700" height="400"
 data="http://releases.flowplayer.org/swf/flowplayer-3.2.16.swf"
 type="application/x-shockwave-flash">
  <param name="movie" value="butterfly.mp4">

  <video controls width="700" height="400">
    <source src="beach.mp4" type="video/mp4">
    <source src="beach.webm" type="video/webm">

    <img src="beach_thumbnail.jpg" alt="A lazy day at the beach">
    <p>Your browser does not support HTML5 video or Flash.</p>
    <p>You can download the video in <a href="beach.mp4">MP4 H.264</a>
    or <a href="beach.webm">WebM</a> format.</p>
  </video>
</object>
```

顺便说一下，有很多JavaScript播放器直接支持HTML5并且有一个内置的Flash后备措施。比如，Flowplayer提供了一个叫Flowplayer HTML5的版本（可以在http://flowplayer.org下载），它会根据需要自动选择使用<video>元素或Flash后备措施。这种方式的优势是简化了HTML代码，因

为有个小能手（JavaScript驱动的媒体播放器）处理了一切；不足之处在于这远离了纯粹的HTML5解决方案，在HTML5浏览器无处不在的那一天，这会是你想立刻就用的方案。

5.5 使用 JavaScript 控制播放器

到现在为止，我们介绍了一些基础知识，也知道了怎么围绕新的<audio>和<video>元素拿出合理的支持方案，使媒体文件不仅能在Flash播放器里播放，而且能在更多的网页中直接播放。这些新技术还是能派上用场的。

如果只谈标记，那么利用<audio>和<video>也就只能做那么多了。然而，这两个元素都有一个扩展的JavaScript对象模型，让我们能通过代码控制播放过程。事实上，甚至还可以调整一些细节，比如播放速度。这些功能是浏览器标准的音频和视频播放器力所不能及的。

好了，接下来的几节，我们通过两个实际的例子，介绍JavaScript对音频和视频的支持。第一个例子是为游戏添加音效，第二个例子将创建一个自定义的视频播放器。最后，我们还将讨论几个其他人利用HTML5和JavaScript开发出来的解决方案，包括可换肤的播放器和无障碍字幕。

5

5.5.1 添加音效

通过<audio>元素并不是只能播歌曲和录音，还可以利用它播放音效。这一点使其特别适合为游戏配乐和添加音效。

图5-6展示了一个非常简单的例子，网页中显示着可交互的弹跳球动画。在第8章学习<canvas>的时候，我们再介绍这个例子的代码。现在，我们需要考虑的是怎么为它配上合适的背景音乐。

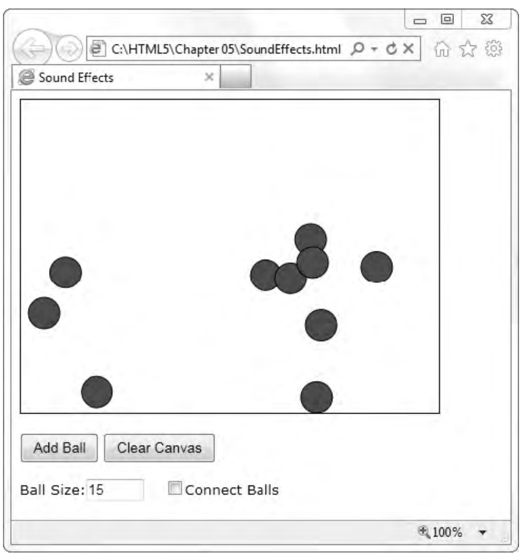


图5-6：这个页面在画布元素上运行一个简单的动画，单击按钮可以添加新的弹跳球（新球会下落，并不断在画布区域内弹跳）。另外，用鼠标单击可以改变球的弹跳方向

实际上，这个例子组合了背景音乐和音效。添加背景音乐很简单，只要在页面中加上一个不可见的<audio>元素即可：

```
<audio id="backgroundMusic" loop>
  <source src="TheOwlNamedOrion.mp3" type="audio/mp3">
  <source src="TheOwlNamedOrion.ogg" type="audio/ogg">
</audio>
```

我们没有给音频播放器添加autoplay或controls属性，因此开始的时候没有声音，也看不到播放器界面。不过，倒是添加了一个loop属性，这样只要一开始播放，就会不断反复播放。为了控制声音的播放，需要使用两个音频（或视频）对象的方法：play()和pause()。怎么没有停止播放的方法呢？的确没有，要停止播放，可以先暂停视频，然后将其currentTime属性重置为0，即下次播放会从头开始。

了解了这些以后，就可以在添加第一个球时开始播放背景音乐了，很简单：

```
var audioElement = document.getElementById("backgroundMusic");
audioElement.play();
```

在清除画布的时候停止播放背景音乐同样很简单：

```
var audioElement = document.getElementById("backgroundMusic");
audioElement.pause();
audioElement.currentTime = 0;
```

前面介绍过，同时播放多少个音频文件是没有限制的。因此，在背景音乐开始播放的同时，我们可以再琢磨一下怎么添加更好玩的音效。

对这个例子来说，小球每次从地面或墙上弹开的时候，都要播放弹簧突然释放的“啵嘤”声。为了增强趣味性，我们使用了几种不同的音效。当然，我们这里只是模仿真实的游戏，如果是真实的游戏，恐怕要集成十几个甚至更多声音文件。

实现上述设计的方案有很多，但并不都可行。比如，可以再添加一个<audio>元素，用于播放音效。然后，每次小球撞击反弹时，都通过修改其src属性加载一个新音频文件。这个方案有两个问题。首先，每个<audio>元素每次只能播放一个声音文件，因此如果有多个球连续反弹，那要么不播放紧接着的、重叠的声音，要么立即停止前一个音效，加载并播放后一个。其次，重新设置src属性会强迫浏览器请求音频文件。虽然有些浏览器能很快加载完新音频（如果音频已经在缓存中了），但IE不行。结果，就是音效会延迟——球都弹开半秒钟了，音效才播放。

更好的方案是使用一组<audio>元素，每个播放一种音效。以下就是示例代码：

```
<audio id="audio1">
  <source src="boing1.mp3" type="audio/mp3">
  <source src="boing1.wav" type="audio/wav">
</audio>
<audio id="audio2">
  <source src="boing2.mp3" type="audio/mp3">
  <source src="boing2.wav" type="audio/wav">
</audio>
<audio id="audio3">
  <source src="boing3.mp3" type="audio/mp3">
```

```
  <source src="boing3.wav" type="audio/wav">
</audio>
```

注意　这里的3个<audio>元素使用了不同的音频文件，但这并不是必须的。比如，你可以使用同一个音频文件，而为了支持重叠音效，可能还得使用3个音频播放器。

发生碰撞反弹时，JavaScript代码会调用一个名为boing()的自定义函数。这个函数会按照顺序取得下一个<audio>元素，然后播放声音。

以下就是实现音效播放的代码：

```
//记录<audio>元素的个数
var audioElementCount = 3;

//记录当前播放的<audio>元素
var audioElementIndex = 1;

function boing() {
  //取得循环列表中的下一个<audio>元素
  var audioElementName = "audio" + audioElementIndex;
  var audio = document.getElementById(audioElementName);

  //播放音效
  audio.currentTime = 0;
  audio.play();

  //把计数器更新为下一个<audio>元素
  if (audioElementIndex == audioElementCount) {
    audioElementIndex = 1;
  }
  else {
    audioElementIndex += 1;
  }
}
```

提示　要体验一下混合了背景音乐和弹跳球音效的页面，可以访问http://prosetech.com/html5。

这个例子能够正常运行，可如果你想实现更大范围的音效怎么办？最简单的方案就是为每个音效单独创建一个隐藏的<audio>元素。如果不行，还可以动态设置已有<audio>元素的src属性。甚至，可以像下面这样动态地创建新<audio>元素：

```
var audio = document.createElement("audio");
audio.src = "newsound.mp3";
```

或者更简单：

```
var audio = new Audio("newsound.mp3");
```

不过，这两种方式都存在问题。首先，必须在播放音频之前设置src属性。否则就会造成音效延迟，在IE中特别明显。其次，必须知道浏览器支持的音频格式，这样才能设置正确的文件类型。为此，就得使用笨拙的canPlayType()方法。给这个方法传入音频或视频的MIME类型，它会告诉你浏览器是否能播放该格式——基本上准确。如果浏览器不支持传入的类型，canPlayType()方法会返回空字符串，如果该方法认为浏览器可以，则返回“probably”，如果它希望浏览器可以，则返回“maybe”，但这些都不能保证真的可以播放。之所以会有判断失误的时候，主要是因为有时候浏览器虽然支持相应的容器，但该容器中却使用了浏览器不支持的编解码器，而即使浏览器支持编解码器，但也可能不支持其编码设置。

大多数开发人员喜欢下面这种编码方式，即只要canPlayType()不返回空字符串，就说明浏览器支持相应的格式：

```
if (audio.canPlayType("audio/ogg")) {
  audio.src = "newsound.ogg";
}
else if (audio.canPlayType("audio/mp3")) {
  audio.src = "newsound.mp3";
}
```

5.5.2　创建自定义视频播放器

使用JavaScript操作<audio>和<video>元素的最常见理由，就是构建自己的播放器。基本思想很简单，不添加controls属性，但有播放窗口，因此可以在下方添加自己的播放控件。最后，添加JavaScript代码，让新控件发挥作用。图5-7展示了一个例子。



图5-7：制作自己的HTML5视频播放器很简单（要做好可不简单）。这个例子中包含标准的播放控件、播放进度条和其他一些按钮，展示了JavaScript对<video>元素的全方位控制能力

每个视频播放器都需要一套基本的播放按钮。图5-7就使用了普通的按钮。

```
<button onclick="play()">Play</button>
<button onclick="pause()">Pause</button>
<button onclick="stop()">Stop</button>
```

这几个按钮会触发以下简单的函数：

```
function play() {
  video.play();
}

function pause() {
  video.pause();
}
function stop() {
  video.pause();
  video.currentTime = 0;
}
```

另外3个播放控制按钮可不太常见，它们可以修改playbackRate以改变播放速度。例如，把playbackRate设置为2，视频会以正常速度的两倍播放，不过由于有音高校正，所以音频听起来正常，只是速度加快了。这个功能对于快速看完一段拖沓的培训视频是很有用的。类似地，把playbackRate设置为0.5，会导致视频比正常速度放慢一半播放，而把playbackRate设置为–1，速度不变，只是会倒退播放——在有些浏览器中，可能无法顺畅地实现此功能：

```
function speedUp() {
  video.play();
  video.playbackRate = 2;
}

function slowDown() {
  video.play();
  video.playbackRate = 0.5;
}

function normalSpeed() {
  video.play();
  video.playbackRate = 1;
}
```

创建播放进度条更有意思一些。进度条的标记实际上就是两个嵌套的<div>元素：

```
<div id="durationBar">
  <div id="positionBar"><span id="displayStatus">Idle.</span></div>
</div>
```

提示 播放进度条是非常适合使用<progress>元素（参见4.6.2节）实现的功能。但是，由于支持<progress>元素的浏览器有限——比支持HTML5视频的浏览器少得多，因此这个例子就使用两个<div>来实现类似的功能。

外面的<div>元素（durationBar）显示实心边框，勾勒出进度条的整个轮廓，表示视频的完整播放时间。内部的<div>元素（positionBar）表示当前播放的时间点，视觉上就是在黑色方框中填入蓝色。最后，内部<div>中的<span>元素保存着状态文本，用于在播放期间显示当前时间点（秒）。

用于为这两个长条添加样式的CSS规则如下：

```
#durationBar {
  border: solid 1px black;
  width: 100%;
  margin-bottom: 5px;
}

#positionBar {
  height: 30px;
  color: white;
  font-weight: bold;
  background: steelblue;
  text-align: center;
}
```

在视频播放过程中，<video>元素会连续触发onTimeUpdate事件，通过响应这个事件可以更新播放进度条：

```
<video id="videoPlayer" ontimeupdate="progressUpdate()">
  <source src="butterfly.mp4" type="video/mp4">
  <source src="butterfly.webm" type="video/webm">
</video>
```

在此，代码会取得视频的当前时间点（保存在currentTime属性中），除以视频总时间（保存在duration属性中），然后将得到的百分比转换为positionBar <div>元素的百分比长度：

```
function progressUpdate() {
  //动态设置蓝色的positionBar，从0到100%
  var positionBar = document.getElementById("positionBar");
  positionBar.style.width = (video.currentTime / video.duration * 100) + "%";

  //显示已经播放的秒数，保留两位小数
  displayStatus.innerHTML = (Math.round(video.currentTime*100)/100) + " sec";
}
```

提示 如果你还想做得更精致一点，可以再添加一个下载进度条，显示当前已经下载和缓冲的多少内容。浏览器已经把这个功能内置在它们自己的播放器中了。如果你要自己做，需要处理onprogress事件，并利用seekable属性。要了解<video>元素提供的更多属性、方法和事件，可以看看微软这篇文章http://tinyurl.com/video-obj-js。

5.5.3 JavaScript媒体播放器

如果你真的想标新立异，你可以自己开发自己的音频和视频播放器。但这可不是一项小工程，

特别是像交互式播放列表这样的时髦功能，不容易实现。而且，如果不是有一个艺术设计部分作后盾，最终出来的产品界面很可能会丑陋不堪。

别苦恼了，除了自己开发，还有很多现成的HTML5播放器可供选择。网上有很多人已经开发了免费的基于JavaScript的媒体播放器。在此，我推荐两个靠谱的，一个是VideoJS（http://videojs.com/），另一个是jQuery粉丝们喜欢的jPlayer（http://www.jplayer.org/）。这两款播放器都很小，使用方便，而且可以换肤。换肤的意思就是通过更换样式表，可以修改播放控件的外观。

大多数JavaScript媒体播放器（包括VideoJS和jPlayer）都内置了Flash后备，这样就省得我们自己去下载Flash播放器了。而且，jPlayer还提供了自己的播放列表功能，让用户可以把音频和视频文件组织到一个列表中（图5-8）。



图5-8：使用jPlayer的播放列表功能，可以提供多个音频或视频文件。用户既可按顺序播放，也可以单击播放任意一个。图中的播放列表包含三个视频文件

5

要使用VideoJS，首先需要从VideoJS网站下载JavaScript文件。然后，像下面这样在页面中添加对相应JavaScript和样式表文件的引用：

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>...</title>
  <script src="video.js"></script>
  <link rel="stylesheet" href="video-js.css">
</head>
...
```

之后，像平常使用<video>元素一样，准备多个<source>元素和Flash后备即可。（VideoJS播放器的示例代码中已经内置了Flowplayer作为Flash后备，如果你不喜欢，可以换成其他Flash播放器。）事实上，要说普通HTML5视频页面与使用VideoJS的页面有什么不一样，那就是对于后者，你必须得用一个特殊的<div>元素包装<video>元素，如下所示：

```
<div class="video-js-box">
  <video class="video-js" width="640" height="264" controls ...>
    ...
  </video>
</div>
```

仅此而已。虽然是扩展HTML5，但这么简单还真是让人惬意。

5.6 视频字幕

正如我们前几章介绍过的，HTML5的制定者经常会考虑Web的无障碍性。所谓无障碍，就是让残疾人也能方便快捷地使用功能丰富的网页。

为图片添加无障碍信息很容易，只要在alt属性中放上一段合适的描述性文本即可。那么，视频流中与alt文本对应那个东西应该是什么呢？人们一致认为是字幕，也就是在视频播放期间，适时出现的一段段的内容对白或话外音。说起字幕，有人可能会想到电视上的闭源字幕（closed caption），通常都是人物对话或者画面内容的补充说明。关键是，字幕可以让即使听不见的人（或者不想在办公室里打开电脑音箱，但又忍不住观看《钢铁侠4》预告片的人）也能把视频看明白。

图5-9是一个有字幕的视频。

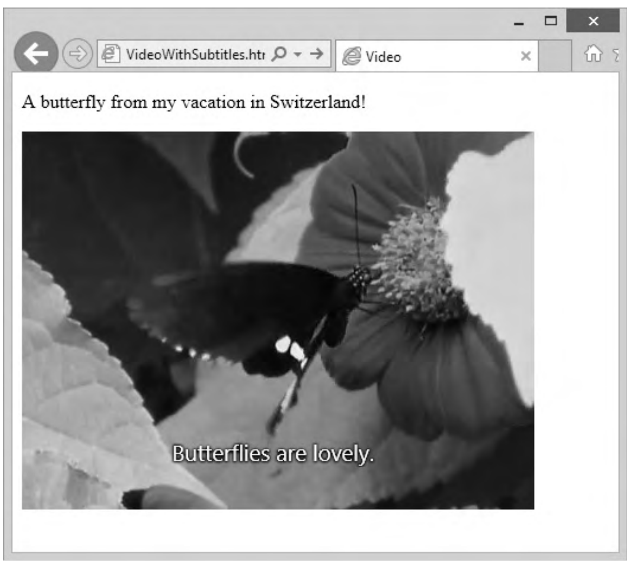


图5-9：这个视频的字幕在常见的位置：视频窗口底部居中的位置

5.6.1 标记时间的文本轨道和WebVTT

从视频技术上来讲，字幕是叠加在视频上的说明文字。一个字幕序列就是标记时间的文本轨道，它有多种格式，但有一些基本的相似之处。它们都是带有时间标记的、以时间顺序排列的普通文本，保存在普通文本文件中。这里有一个以WebVTT（Web Video Text Tracks）格式写的时间标记文本轨道，这是HTML5偏好的格式。下面是4行字幕：

```
WEBVTT

00:00:05.000 --> 00:00:10.000
This caption appears 5 seconds in and lingers until the 10 second mark.

00:01:00.000 --> 00:01:10.000
Now 1 minute has passed. Think about that for 10 seconds.

00:01:10.000 --> 00:01:15.000
This caption appears immediately after the second caption disappears.

00:01:30.000 --> 00:01:35.000
Captions can use <i>line breaks</i> and <b>simple</b> HTML markup.
```

如你所见，WebVTT文件的每一条都指明了3项信息：字幕出现的时间（以小时：分钟：秒的格式）、字幕消失的时间和相关字幕文本。把这些内容保存在一个扩展名为.vtt（比如subtitles.vtt）的文件中，就得到了一个随时可用的标记时间文本轨道文件。

虽然字幕看上去简单，却有许多要求高精度的细节。比如，你可能想要控制换行，格式化文本，将字幕移到视频窗口的其他位置，或者显示一次填充一个字的卡拉OK格式字幕。这就是为什么有将近50种不同的标记文本轨道格式。事实上，时间标记文本标准之间的斗争和音视频编码格式战争一样丑恶不堪。

目前，HTML5正式标准并没有指定应该用哪种格式。但是，浏览器厂商一致支持WebVTT这个仍在演变中的标准，它是受桌面媒体播放器所使用的SRT格式启发演变而来。浏览器厂商没有采用W3C已经打磨了10年的更成熟的TTML（Timed Text Markup Language）标准，因为它太复杂了。目前，IE10（以及更新版）是唯一全方位支持TTML的浏览器。

提示 可以在http://dev.w3.org/html5/webvtt了解更多关于WebVTT标准的知识，包括修改字幕格式和样式的技术。如果需要写字幕方面的帮助，可以访问微软漂亮的字幕生成器页面（如图5-10），地址是http://tinyurl.com/capmaker。



图5-10：要使用字幕生成器，首先必须给它指定一个已经在Web上的视频文件（输入URL然后点击Load）。在视频播放时，快速添加字幕很容易（在视频窗口下面的框中输入，然后点击Save）。当一切都做完后，可以从下面的文本框中复制出完整的WebVTT文本

5.6.2 使用<track>添加字幕

拥有WebVTT字幕文件之后，就需要将其与视频文件配对。使用的元素是<track>。将其添加到<video>元素内，在任意<source>元素之后：

```
<video controls width="700" height="400">
  <source src="butterfly.mp4" type="video/mp4">
  <source src="butterfly.webm" type="video/webm">
  <track src="butterfly.vtt" srclang="en" kind="subtitles" label="English"
   default>
</video>
```

<track>元素有几个属性。首先是src属性，指定了字幕文件。srclang属性指定了字幕文件的语言代码，这是给无障碍访问工具准备的。这里用en表示语言为英文（可以从http://tinyurl.com/l-codes中获取更具异域风情的语言代码）。

kind属性描述了字幕中的内容类型。HTML5标准提供了5种选择，但只有两种可用作弹出式字幕。如果字幕中包含了对话的转录或翻译，可以设置subtitles属性值；如果字幕中包含了对

话和音效及音乐线索的描述，可以设置captions属性。

提示　当你能听见音频却不懂它的意思时，用subtitles合适，比如，在看一个外语片的时候。当根本没有音频的时候，用captains合适，比如，为了不吵醒旁边工位的同事而将播放器静音的时候。

Kind属性的值还有descriptions（无法获取视频时的替换文本，并且可以被语音合成器读出来）、chapters（章节标题，可用于导航）、metadata（可以通过JavaScript代码获取的少量信息）。视频播放器不会显示这些属性的值，所以需要另一个工具或者JavaScript代码来获取并使用这些信息。

Label属性用来设置显示在视频播放器字幕菜单中的文本，点击视频窗口下面的小按钮可以调出这个菜单。如果想要让用户从多个字幕文件中选择，标签文本就非常重要了。比如，下面这个视频有两个字幕文件：


```
<video controls width="700" height="400">
  <source src="butterfly.mp4" type="video/mp4">
  <source src="butterfly.webm" type="video/webm">
  <track src="butterfly.vtt" srclang="en" kind="subtitles" label="English"
   default>
  <track src="butterfly_fr.vtt" srclang="fr" kind="subtitles" label="French">
</video>
```

第一个<track>元素有default属性，所以它就是初始化时会选用的字幕。但用户可以点击字幕按钮选择另一个字幕文件（如图5-11）。

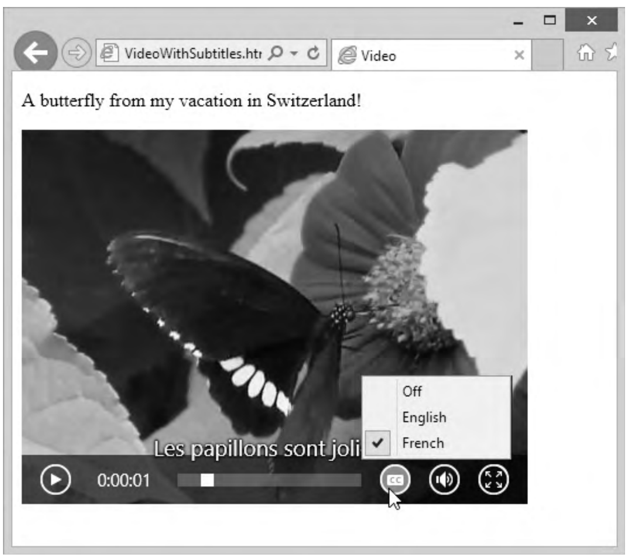


图5-11：支持<track>元素的浏览器会像左图这样添加一个字幕选择按钮。有了它，用户可以切换或者关闭字幕

即便视频只有一个字幕文件，字幕列表仍然有两个选择：字幕和一个关闭字幕的选项。如果想让用户可以选字幕,但希望字幕初始时是关闭的,只需确保没有任何<track>元素设置了default属性。这样，视频在开始播放时就关闭了字幕。

注意 字幕文件不只是为了无障碍访问和无声播放。搜索引擎也可以抓取字幕文件里的信息并用它改善搜索结果。事实上，未来的超级智能搜索引擎可以用WebVTT信息，通过匹配检索关键字和字幕，让检索者直接到达视频中指定的播放位置。

5.6.3 浏览器对视频字幕的支持情况

浏览器渐渐开始支持<track>元素。在写这本书的时候，Firefox还完全不支持，但Mozilla的开发者们计划将来支持。表5-4列出了当前的支持情况。

表5-4 浏览器对<track>元素的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android
最小版本号	10	–	26	6*	15	–	2

* Safari没有提供用于切换和打开或关闭字幕的按钮。

注意 如果在Chrome中测试使用字幕的视频，需要将视频文件上传到服务器。如果只是简单地从本地启动视频文件，Chrome可以播放视频，但不能抓取匹配的WebVTT文件。

幸运的是，可以放心使用<track>元素。不支持的浏览器会忽略它，没有不良影响。

如果需要在所有HTML5浏览器均有效的字幕方案，有一种简单的变通方案。可以如往常一样使用<track>元素，再结合JavaScript作为补充，比如Captionator.js（http://captionatorjs.com）。Captionator会在视频窗口上覆盖一个浮动元素，当播放到合适的点，Captionator.js会从WebVTT文件中获取字幕文本并将其插入到浮动元素中。

第 6 章

美妙的CSS3字体和特效

制作现代网站，如果不使用CSS3，是不可能的。作为Web标准，CSS3已经像HTML一样，成为了制作网页或开发Web应用不可或缺的一个部分。不论是布局页面、构建交互按钮和菜单，还是仅仅美化一下界面，CSS都是最基本的工具。事实上，随着HTML慢慢地将关注点转向内容和语义（参见2.1节），CSS已经成为Web设计的灵魂所在。

作为Web设计的核心语言，CSS变得日益庞大和复杂。到了CSS 2.1的时候，CSS规范扩展为最初的5倍，几乎达到中篇小说的篇幅。好在，CSS的设计者们对这个标准有着合理的长远规划。他们把下一代CSS拆分为一组独立的标准，每个独立的标准叫做模块。这样一来，浏览器开发商就可以自主决定先实现哪个模块，从而让那些令人向往的部分尽早得到应用。而且，浏览器开发商确实也是这么做的，要么实现某个模块，要么不实现。所有新的CSS模块归总起来叫做CSS3（注意，跟HTML5一样，字母与数字之间没有空格）。

CSS3在不断发展成熟的过程中出现大约50个模块。这些模块涵盖了悦人眼目的功能（比如丰富的字体和动画），也包括更具体、更有针对性的内容（如朗读文本和根据计算机或移动设备改变样式的能力）。总之，在这众多的模块中，有些已经得到当前所有浏览器最新版本的支持，而有些则只是尚未得到任何浏览器支持的试验性规范。

本章将讨论CSS3最重要的部分（也是被支持得最好的部分）。先看一看怎么使用字体让页面文本更加活泼，然后再探讨如何编写样式以适应不同大小的浏览器窗口和不同的上网设备（如iPad和iPhone）。接下来，我们会看看怎么使用阴影、圆角和其他手段把方框变得更漂亮。最后，再讨论一下怎么使用渐变创建精妙的效果，在鼠标悬停、单击或切换控件时，给出优雅的视觉提示。（结合变换和透明这两个CSS3功能可以把上述效果做得更完美。）最终，我们将学习如何用丰富多彩的网络字体使普通文本生动起来。

不过，在着手尝试这些激动人心的CSS3功能之前，我想有必要先考虑下，怎样才能做到既为网页应用拉风的特效，同时又不会把一部分用户抛在滚滚风尘中。

6.1　使用 CSS3

毋庸置疑，CSS3是Web样式设计的未来，但它还没有制定完成。许多模块都在修订和评审当中，没有一款浏览器支持全部模块。可以在http://tinyurl.com/CSS3-stages查看这个庞大的标准家

族当前的状态。

因为CSS3还在微调中，所以它与HTML5一样，都存在兼容性问题。因此，每一位网站开发者都要自己决定使用什么，不使用什么，以及如何做好不同用户体验之间的衔接工作。

在打算在网站中使用CSS3之前，你有三个选择。下面我就一一分析给你听。

注意 CSS3不是HTML5的一个部分。这两个标准是独立的，制定它们是两批人，时间、地点也都不一样。然而，即使W3C都在鼓励开发人员把HTML5和CSS3混同为一个标准，希望借此推动Web的新一波发展浪潮。不相信？看看W3C的HTML5标志生成页面：http://www.w3.org/html/logo，你会发现W3C鼓励你在HTML5的标志中宣传CSS3。此外，许多基于HTML5的现代Web设计（比如我们将在下一章学习的适用于手机的布局技术）都有一个显著特征——需要CSS3。

6.1.1 选择一：用能用的

如果某个功能得到了所有浏览器的支持，自然就可以放心使用。Web字体就是这样的功能（参见6.4节）你大可以选择合适的字体，这些字体甚至都可以在IE6中显示。可惜的是，这样的CSS3功能实在是少之又少。另外，单词自动折行（`word-wrap`属性）也能在所有浏览器中使用，而稍作处理就可以在老版本浏览器中实现透明效果。可是，除了这些之外，几乎所有功能对现在依然流行的IE8而言都是不兼容的。

注意 除非特意说明，本章所有CSS3功能都可以在目前最新版本的浏览器中使用，包括IE，如果你用的是IE9及其更新版本的话，但不包括其他老版本的IE。

6.1.2 选择二：将CSS功能作为增强

CSS3粉丝们异口同声地说："网站没有必要在所有浏览器中都长得一模一样。"当然没错。（他们还为此建了一个单页网站呢：http://DoWebsitesNeedToBeExperiencedExactlyTheSameInEveryBrowser.com/，它针对现代浏览器做了一些装饰，但对像IE7这样落后的浏览器只保留了基本功能。）

这个选择的中心思想，就是利用CSS3来添加装饰，即使用户的浏览器不支持也无伤大雅。比如说`border-radius`属性吧，可以用它生成浮动的圆角框：

```
header {
  background-color: #7695FE;
  border: thin #336699 solid;
  padding: 10px;
  margin: 10px;
  text-align: center;
  border-radius: 25px;
}
```

支持`border-radius`属性的浏览器知道该怎么做。而老版本浏览器则会忽略这条声明，仍然呈现方角框（图6-1）。

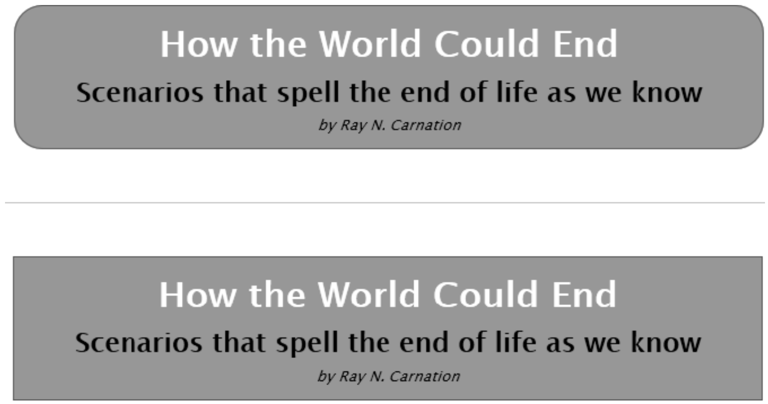


图6-1：在IE9中，标题框有圆角（上）。IE8则会忽略`border-radius`属性，只应用其他的样式声明（下）

这种向后兼容让设计师可以在支持最新版CSS的浏览器上使用最新潮的设计效果，而在旧版浏览器上网站也不会被破坏。不过，如果你在这条路上走得太远恐怕还是有问题。一个网站，不管它在最新版本的浏览器中看起来有多好，只要有一部分用户在用旧浏览器，而且在他们的浏览器中看不出你的网站有什么特别，那你的投入产出就要大打折扣。毕竟，谁都希望自己的网站能吸引所有人，而不只是那些拥有最新浏览器的极客。

考虑到这一点，在使用某些CSS3功能增强网页时，不得不多加小心。换句话说，应该只使用那些已经得到大多数浏览器（即便是需要最新版）支持的功能。不要让自己的网站在不同浏览器中的体验差别过大，以免强迫部分用户受到“低一等”的待遇。

6

提示 一说到CSS3，IE总是那个“掉队”的家伙。少数激进的Web设计师甚至主张应该把像IE8这种落后的浏览器“打入冷宫”，只要其他浏览器支持某个CSS3功能，就可以使用，而不必考虑IE。否则，又有谁能给微软施压，让Web发展得越来越好呢？从理论上讲，这样做没有问题，但仅限于你的网站本身致力于推广先进的Web标准。如果不是的话，那么拒绝一大部分用户的结果只能适得其反。道理很简单，你可以不喜欢别人的浏览器，但别人却会用自己的浏览器来浏览**你的**工作成果。

6.1.3 选择三：Modernizr

如果你使用了某个未得到全面支持的CSS3功能，同时不支持该功能的浏览器也能给出不错的呈现效果，那当然再好不过了。可是，有时候这样做会导致网站中某个关键的区块消失不见，或者其降级的版本看起来丑陋不堪。比如，图6-2展示了只有Firefox支持的多色边框。

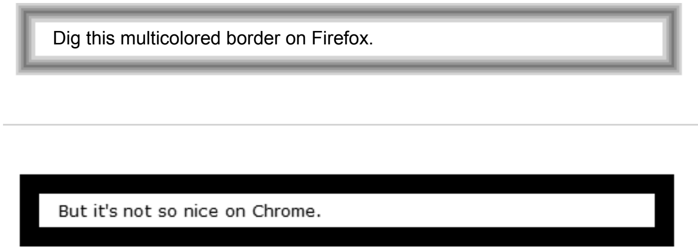


图6-2：在Firefox中，多色边框看起来赏心悦目（上）。但在Chrome中，我们只能看到傻粗傻粗的纯黑色边框（下），这怎么说也不能叫好看

对这种问题，有时候可以通过按正确的顺序排列属性声明来解决。换句话说，就是先列出兼容性最好的属性，然后再列出新属性，以覆盖之前的属性。这种方式如果有效，那么就可以适应所有浏览器。因为老版本浏览器会采用标准的属性，而新浏览器则会用新属性覆盖标准的属性。举个例子吧，可以利用属性覆盖技术实现用渐变背景代替实色背景：

```
.stylishBox {
  ...
  background: yellow;
  background: radial-gradient(ellipse, red, yellow);
}
```

图6-3展示了结果。

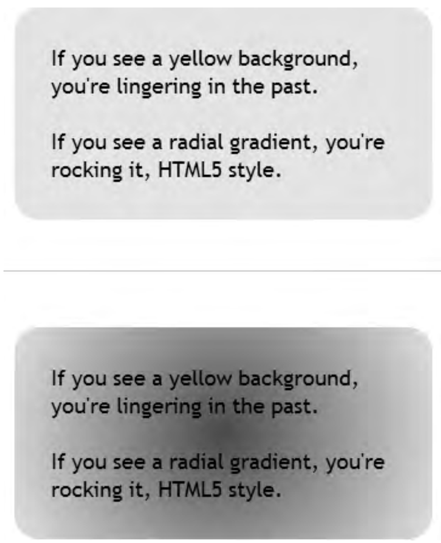


图6-3：上：在不支持CSS3的浏览器中，这个类名为stylisBox的元素只有黄色背景。下：在支持CSS3的浏览器中，黄色背景会被由红到黄的放射性渐变取代

在某些情况下，覆盖样式属性也不能奏效，因为需要以组合的方式来设置属性。图6-2所示的多色边框就是一个例子。设置这个多色效果要使用`border-colors`属性，但看起来就好像是只通过`border-thickness`属性把边框加粗了一样。在不支持多色边框的浏览器中，那个傻粗傻粗的边框怎么看怎么难看，不管它是什么颜色。

要解决这个问题，可以使用Modernizr，也就是1.6.7节介绍的那个用于测试HTML5功能支持的JavaScript库。使用Modernizr可以为不支持某个样式属性的浏览器设置替代的样式。例如，假设要创建图6-1所示的两种标题框。在支持的浏览器中，你想要圆角边框，而在不支持的浏览器中，你想使用双线边框。此时，如果你在页面中引用了Modernizr脚本，那么可以像下面这样写出组合的样式规则：

```
/*为所有标题设置样式，无论浏览器是否支持CSS3*/
header {
  background-color: #7695FE;
  padding: 10px;
  margin: 10px;
  text-align: center;
}

/*为支持border-radius属性的浏览器设置样式*/
.borderradius header {
  border: thin #336699 solid;
  border-radius: 25px;
}

/*为不支持border-radius属性的浏览器设置样式*/
.no-borderradius header {
  border: 5px #336699 double;
}
```

这个办法不错啊，那两个类是什么意思呢？在页面中使用Modernizr时，需要给页面的根元素`<html>`添加`class="no-js"`属性：

```
<html class="no-js">
```

然而，在页面加载完Modernizr后，它会迅速检测一批HTML5、JavaScript和CSS3功能的支持情况。然后，为`<html>`元素应用一大堆类，每个类名用空格隔开，把其`class`属性修改成如下所示：

```
<html class="js flexbox canvas canvastext webgl no-touch geolocation
postmessage no-websqldatabase indexeddb hashchange history draganddrop
no-websockets rgba hsla multiplebgs backgroundsize borderimage borderradius
boxshadow textshadow opacity no-cssanimations csscolumns cssgradients
no-cssreflections csstransforms no-csstransforms3d csstransitions fontface
generatedcontent video audio localstorage sessionstorage webworkers
applicationcache svg inlinesvg smil svgclippaths">
```

如果这里列出的类名中包含某个功能，说明浏览器支持该功能。如果表示相应功能的类名前缀为“no-”，那说明浏览器不支持该功能。以上面的代码为例，说明浏览器支持JavaScript（`js`），但不支持Web套接字（`no-websockets`）。至于CSS3，说明浏览器支持`border-radius`属性（`borderradius`），但不支持CSS3倒影（`no-cssreflections`）。

在自己的选择符中使用这些类名，就可以基于浏览器的支持情况分别设置样式。比如，使用选择符`.borderradius header`能够取得所有包含在这个`<html>`元素中的`<header>`元素——当然，浏览器必须得支持`border-radius`属性。要不然，就不会出现`.borderradius`类，而这个选择符也就不会选择任何元素，对应的规则也会被忽略。

使用Modernizr也有一个问题，那就是它只会检测一部分CSS3功能。这部分功能都是最流行也最成熟的CSS3功能，其中并不包括图6-2所示的`border-color`功能，因为相应的属性还只有Firefox自己支持。为此，最好还是不要在自己的网页中使用多色边框，至少现在不行。

注意 利用Modernizr也可以编写作为后备（向后兼容）的JavaScript脚本。编写脚本时，只要像检测HTML5功能一样，检测Modernizr对象中的特定属性即可。在浏览器不支持的高级CSS3功能太多（比如不支持渐变和动画）的情况下，也可以用编写脚本的方式来弥补。不过，这样一来工作量可就大了，而且实现方式也完全不同。所以对于一些必不可少的网站功能，最好只通过JavaScript来实现。

6.1.4 有开发商前缀的特定于浏览器的样式

制定CSS标准的人在引入新功能时，经常会遭遇“蛋和鸡”的困局。为了让一项功能臻于完美，他们需要听到浏览器开发商和Web设计人员的反馈。但是，为了得到这些反馈，必须先让浏览器开发商和Web设计人员实现还不够完美的功能。这样就会形成一个试验和反馈循环，经过反复多次修订，最终定案。在此期间，无论是功能的语法还是实现，都会发生变化。于是不可避免地会导致非常现实的风险：某些Web设计人员会学习这些新功能，然后将其用在自己的网站中，而将来标准万一有变化，就可能导致网站无法使用。

为了避免这种风险，浏览器开发商使用了一种叫做*开发商前缀*（vendor prefix）的办法，即为还在开发中的CSS属性和功能加上特定浏览器实现的前缀。比如6.2.5节将要介绍的还在开发过程中的`radial-gradient()`函数。在旧版的Firefox中还没有`radial-gradient()`函数，但是可以使用名为`-moz-radial- gradient`的方法。

```
.stylishBox {
  background: yellow;
  background: -moz-radial-gradient(ellipse, red, yellow);
}
```

Firefox所使用的开发商前缀是`-moz-`（开发Firefox的组织Mozilla的缩写）。每个浏览器引擎都有自己的开发商前缀（见表6-1）。虽然看起来是人为制造了不少麻烦，但这样还是有其合理性的。首先，不同浏览器开发商不会同时支持某项功能，而且经常会实现同一规范的不同版本。其次，尽管大家将来都要支持最终规范规定的同样的语法，但特定于浏览器的属性和功能则不一定相同。

表6-1 开发商前缀

前　缀	浏　览　器
`-moz-`	Firefox
`-webkit-`	Chrome、Safari和最新版的Opera（3个浏览器的引擎都是WebKit）
`-ms-`	Internet Explorer
`-o-`	旧版的Opera（15之前）

下面这个例子用了全部4个浏览器特定前缀实现放射性渐变：

```
.stylishBox {
  background: yellow;
  background-image: -moz-radial-gradient(circle, green, yellow);
  background-image: -webkit-radial-gradient(circle, green, yellow);
  background-image: -o-radial-gradient(circle, green, yellow);
  background-image: -ms-radial-gradient(circle, green, yellow);
}
```

显然，在处理这些不太成熟的CSS3时，需要一些“浮肿”的样式规则。

一个明显的问题摆在每个Web设计人员面前：“什么时候需要用开发商前缀？什么时候使用普通的、没有前缀的属性或方法名是安全的？”可能你会考虑给浏览器做一个快速测试，但除非你在每个浏览器上都进行测试，否则无法得到确定性的结论。比如，所有浏览器都支持`borderadius`属性，不需要开发商前缀。但是`radial-gradient()`却有点棘手：在写这本书的时候，大部分浏览器都支持，但一些移动浏览器仍然需要`-webkit-`前缀。还有我们将在本章后面用到的`transform`属性，IE和Firefox浏览器不需要前缀，但Chrome、Safari和Opera仍然需要`-webkit-`前缀。

更麻烦的是，用于指定一个属性值或方法参数的语法可能会变。比如，IE10在测试阶段引入了有前缀的`radial-gradient()`函数。最终，IE10的发布版让开发者既可以用最新的无前缀`radial-gradient()`函数，又可以用稍旧一点的有前缀的`-ms-radial-gradient()`函数。这种设置着实增加了无尽的排错乐趣，呵呵！

6

注意 在本章中，我们将了解到所有涉及的CSS3部分当前的支持情况，包括哪些需要开发商前缀。如果本章例子中的样式**没有**使用开发商前缀，可以认为在页面中使用这些特性时，省略前缀是安全的。

如果现在开始头疼了，别担心，你很容易获取帮助。要获取关于哪些CSS3特性需要开发商前缀的最新信息，可以求助于不可或缺的网站http://caniuse.com（我们第一次在1.6.1节中见过）。当查找一个CSS3特性时，“Can I use…”网站会清楚地指出哪个版本的浏览器需要开发商前缀（图6-4）。

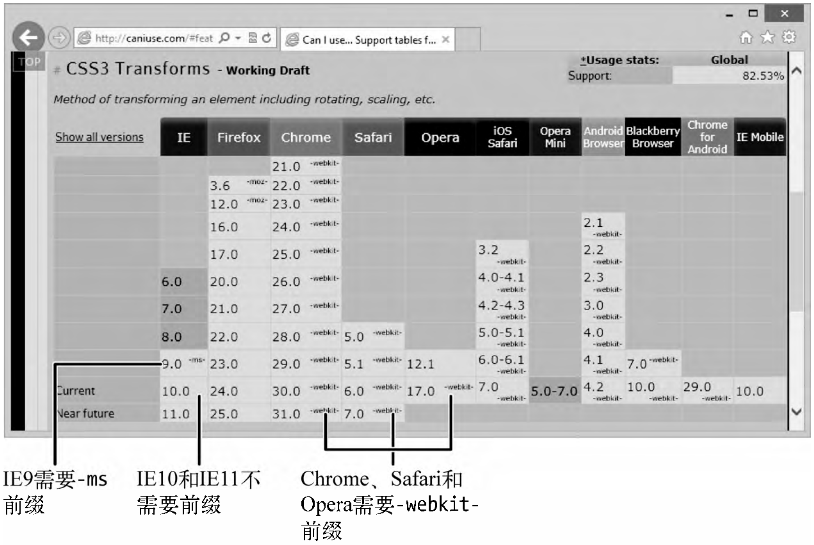


图6-4：如果需要使用前缀，“Can I use…”网站会用一个黄色的文本小气泡表示。比如，CSS3 `transforms`特性在最新的Firefox或IE中不需要前缀，但在其他浏览器中都需要

注意 使用开发商前缀的确是够乱的。Web开发人员也因此分成两个阵营，一个阵营认为要想用上最新最酷的功能，这是无论如何也无法避免的。另一个阵营则认为，那么多浏览器前缀只会让头脑清楚的开发人员对这些功能敬而远之。但有一条是肯定的：如果不使用开发商前缀，相当一部分CSS3功能都将无法使用。

6.2 多变的盒子

从CSS诞生之初，Web设计人员就用它来创建内容盒子。随着CSS的功能越来越丰富，盒子的形式也越来越吸引人，能够用它实现带阴影的标题、浮动的标题插图等很多有用的元素。而当CSS解决了悬停的问题之后，浮动盒子甚至变身成了多种多样、流光溢彩的按钮，一举取代了过去笨拙的JavaScript手段。了解了这些之后，再运用最受欢迎也最受支持的CSS3功能做出更漂亮的盒子——无论用于包含什么内容，也就不足为奇了。

自动添加开发商前缀

如果你大量地使用需要开发商前缀的 CSS3 特性，很快就会疲于更新大量的样式和一遍一遍地给同一个样式添加多个版本。在你崩溃之前，不妨考虑一下神奇卓越的 JavaScript 工具 `-prefix-free`。

要使用`-prefix-free`，只需用你需要的 CSS3 属性创建一个普通的样式，而不需担心开发商前缀问题。然后，在页面中引用`-prefix-free` 脚本。

当用户浏览页面时，-prefix-free 脚本就开始发挥作用。它会检测当前浏览器并自动添加浏览器所需的所有开发商前缀。（没错，这种自动调整会耗费一点额外的时间，但它非常快以至于基本看不出来。）当然，-prefix-free 不是让浏览器支持它本就不支持的特性，而是将普通的、明确命名的属性转变成一些浏览器需要的散乱的、开发商特定的命名，以支持新的和发展中的特性。对许多开发者来说，添加一个 JavaScript 文件来处理 CSS3 前缀的混乱，这个代价还是比较小的。

要下载-prefix-free库，或者要试一下脚本添加CSS前缀的能力，可以访问http://leaverou.github.io/prefixfree。

6.2.1 透明

生成部分透明的图片和颜色是CSS3的一个基本功能。实现透明效果的方法有两种。

第一种是使用rgba()函数，它接收4个数值作为参数。前三个值分别代表色彩中的红、绿、蓝分量，取值范围为0～255。最后一个值是alpha（不透明度）值，取值范围为0～1；0表示完全透明，1表示完全不透明。

下面这个例子会创建一个半透明的黄绿色背景：

```
.semitransparentBox {
  background: rgba(170,240,0,0.5);
}
```

不支持rgba()函数的浏览器会忽略这条规则，而相应的元素会带有默认不透明的背景。当然，更好的做法是先声明一个实心的后备颜色，然后再用半透明颜色覆盖它：

```
.semitransparentBox {
  background: rgb(170,240,0);
  background: rgba(170,240,0,0.5);
}
```

这样，即使浏览器不支持rgba()函数，也会应用指定的背景颜色，只不过完全不透明而已。

提示 为了保证后备背景的效果，尽量选择一种与半透明效果接近的颜色。比如，前面例子中用的是半透明的黄绿色，而下方元素的背景则是白色。由于白色会透出那些半透明的像素，结果就会显得比较亮。在选择后备颜色时，应该尽量选择一个接近的亮色。

另外，CSS3还新增了一个名为opacity的属性，这个属性的原理与alpha值一样（opacity这个单词的意思就是“不透明”）：取值范围为0～1；0表示完全透明，1表示完全不透明：

```
.semitransparentBox {
  background: rgb(170,240,0);
  opacity: 0.5;
}
```

图6-5展示了两个半透明的例子，一个使用rgba()函数，另一个使用opacity属性。

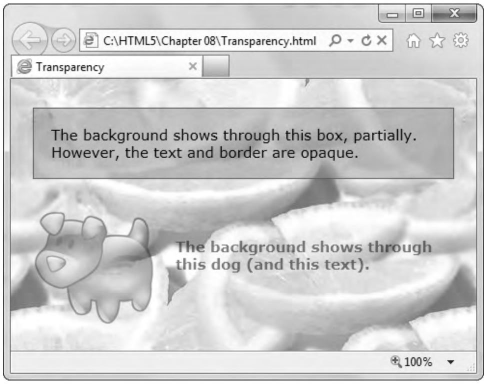


图6-5：这个页面展示了实现半透明效果的两种方式：淡化图片（使用opacity属性）和让背景透过盒子（使用rgba()函数创建半透明的背景色）

在下列情况下，建议使用opacity属性而不是rgba()函数。

- 实现多种颜色（元素）的半透明效果。使用opacity属性，不仅背景颜色，就连文本颜色、边框颜色也会变透明。
- 在不知道颜色的情况下，实现半透明效果（比如，通过其他样式表或者JavaScript代码来设置半透明效果）。
- 实现图片的半透明效果。
- 实现渐变动画效果时，比如元素的淡入淡出（参见6.3.3节）。

6.2.2 圆角

还记得border-radius属性吗？这个属性可以帮我们把盒子的方角刮得圆滑一些，这一点我们介绍过了。但我们没有介绍的是，如何利用这个属性做出曲线来。

首先，可以为border-radius属性选择不同的值，因为这里的radius（半径）指的是圆角的半径。（最终结果不会显示一个完整的圆，只有水平线和垂直线这两条切线以及部分圆弧。）半径值越大，圆弧越长，圆角就越平滑。与CSS中的其他属性类似，这个属性的值也可以使用多种单位，包括像素和百分比。还可以提供4个值，分别对应4个圆角：

```
.roundedBox {
  background: yellow;
  border-radius: 25px 50px 25px 85px;
}
```

不仅如此，还可以把圆拉伸成椭圆，即让某一个轴向的圆弧更长。要实现这个效果，就要单独设定每一个角（使用border-top-left-radius这样的属性），然后提供两个值：一个是水平半径，另一个是垂直半径。

```
.roundedBox {
  background: yellow;
  border-top-left-radius: 150px 30px;
```

```
  border-top-right-radius: 150px 30px;
}
```

图6-6展示了一些例子。

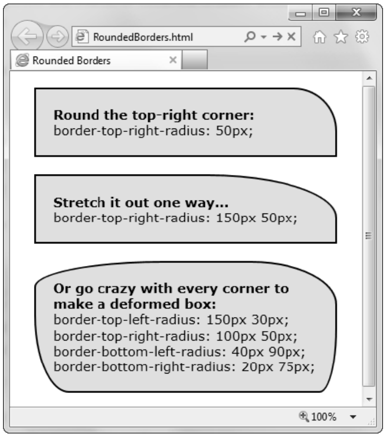


图6-6：动动脑子，利用border-radius属性可以创建出形状各异的盒子

6.2.3 背景

过去，要想设计出带新奇背景和边框的盒子，一般都要求助于图片。CSS3为简化这个任务引入了两个新功能。首先，就是支持多背景，即让你能在一个元素上应用两个甚至更多背景图片。下面这个例子就为一个盒子的左上和右下分别应用了背景：

```
.decoratedBox {
  margin: 50px;
  padding: 20px;
  background-image: url('top-left.png'), url('bottom-right.png');
  background-position: left top, right bottom;
  background-repeat: no-repeat, no-repeat;
}
```

第一步是设定任何数量的图片，使用的还是background-image属性。然后，再设置每张图片的位置并控制它们是否重复，分别使用background-position和background-repeat属性。这里关键是这三个属性值的顺序要匹配。换句话说，第一张图片的位置由background-position属性的第一个值决定，第二张图片的位置由background-position属性的第二个值决定，以此类推。图6-7显示了这个例子的结果。



图6-7：无论盒子有多大多没有关系，背景图片始终都会靠在角上

注意 如果浏览器不支持多背景，那么它会完全忽略这些背景属性。为避免全有全无的结果，最好还是先用background或background-image属性设置一个后备的背景颜色或图片。然后再继续用background-image设置多张图片。

下面这个例子对前面的例子稍微做了一点改进，就实现了原来极费时间的滑动门技术。同样，还是使用了三张图片：一张在左端，一张在右端，还是一张窄窄的在中间：

```
.decoratedBox {
  margin: 50px;
  padding: 20px;
  background-image: url('left.png'), url('middle.png'), url('right.png');
  background-position: left top, left top, right bottom;
  background-repeat: no-repeat, repeat-x, no-repeat;
}
```

这样，就利用多背景创建了一个可以自由缩放的按钮。当然，既然CSS3还提供了更多功能，那么为按钮加上阴影、渐变和其他不依赖图片的效果可能更有吸引力。

6.2.4 阴影

CSS3定义了两种阴影：盒子阴影和文本阴影。这两者中，盒子阴影更有用。比如，可以为

<div>元素添加一个矩形的阴影（同时也要设置边框，这样就仍然有盒子的感觉）。另外，阴影还可以随着盒子的形状而变化（见图6-8）。



图6-8：阴影可以让文本显得浮起来（上）、让盒子突出显示（中）、让按钮产生发光效果（下）

生成这两种阴影的属性分别是box-shadow和text-shadow。下面是创建盒子阴影的一个简单示例：

```
.shadowedBox {
  border: thin #336699 solid;
  border-radius: 25px;
  box-shadow: 5px 5px 10px gray;
}
```

前两个值是水平和垂直方向的偏移量。如果是正值（像这里两个值都是5像素），那么阴影就会向右向下偏移。接下来设置模糊距离——这个例子中是10像素——这增加了阴影的模糊程度。最后一个值是阴影颜色。假如盒子底下有内容的话，可以考虑在这个值的位置上使用rgba()函数（参见6.2.1节），从而将阴影颜色设置为半透明的。

假如你还想再调整一下阴影，还有两个地方可以下手。第一个地方就是在模糊值与颜色值之间，在这个地方加一个值，用于设置阴影伸展范围（spread），即增大模糊边界之前的实心颜色面积：

```
box-shadow: 5px 5px 10px 5px gray;
```

第二个地方就是值列表的末尾，可以在最后加上单词inset，这样就可以在元素内部（而不是外部）创建一个阴影。在不设置水平和垂直偏移值的情况下这个效果是最好的：

```
box-shadow: 0px 0px 20px lime inset;
```

这行代码得到的就是图6-8下面的那个按钮效果。6.3.1节将介绍为按钮添加悬停时的内阴影效果。

注意 我们甚至可以为一个元素添加多个阴影，只要把每个阴影的值列表用逗号分隔开即可。但这种阴影控通常是在浪费时间和电能。

类似地，text-shadow属性也需要一个值列表，但值的顺序有所不同。首先要指定颜色值，然后才是水平和垂直偏移值，最后是模糊值：

```
.textShadow {
  font-size: 30px;
  font-weight: bold;
  text-shadow: gray 10px 10px 7px;
}
```

旧版IE浏览器不支持盒子阴影和文本阴影。盒子阴影需要IE9及其更新版本，而文本阴影需要IE10及其更新版本。

6.2.5 渐变

渐变是多种颜色混合而成的效果，可以用来创建菜单栏后面精细的底纹，或者五彩斑斓的按钮（就像20世纪60年代复活节晚会上的彩灯）。图6-9展示了几个例子。

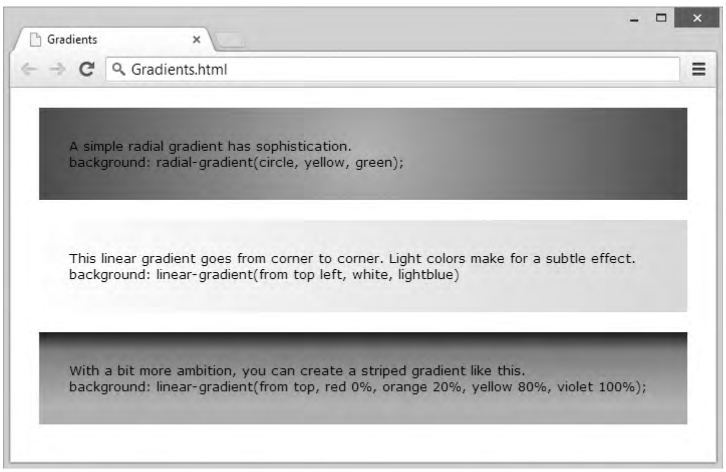


图6-9：本质上，渐变就是两种或更多种颜色的混合。但就是这么简单的拼合，却可以创造出光怪陆离的效果

注意 很多网页中的渐变其实都是用背景图片来伪造的。CSS3可以让我们自己来定义渐变，然后浏览器就会负责呈现。CSS3渐变的优点是可以让浏览器少下载图片，而且这种渐变能够无缝地适应各种大小的空间。

CSS3支持两种渐变：线性渐变和放射性渐变。线性渐变就是沿直线混合几种颜色，而放射性渐变则是在圆心到圆周之间混合颜色。

CSS3并没能给创建渐变提供任何属性。要创建渐变，必须使用渐变函数来设置background属性。提个醒儿，考虑到有的浏览器（比如IE10之前的版本）可能不支持渐变，千万别忘了先写一条background声明，为该属性指定一个实心颜色作为后备。

1. 线性渐变

有4个渐变函数，第一个函数是linear-gradient()。下面的代码展示了使用这个函数的简单形式，这样会创建一个从上到下，从白到蓝的渐变效果：

```
.colorBlendBox {
  background: linear-gradient(from top, white, blue);
}
```

from关键字指定了第一个渐变颜色（白色）的起始位置是顶部（top），可以把它换成to，这样渐变就反转为从底部的蓝色到顶部的白色：

```
background: linear-gradient(to top, white, blue);
```

类似地，将top修改为left就可以创建从一边到另一边的渐变。或者两个都用，创建从左上角开始的对角线渐变：

```
background: linear-gradient(from top left, white, lightblue)
```

如果想多用几种颜色，没问题，只要依次列出颜色值即可。比如，要创建一个红、橙、黄的三色渐变，以红色从顶部开始，这样写就可以了：

```
background: linear-gradient(from top, red, orange, yellow);
```

最后，还可以使用*渐变点*（gradient stop）控制每个颜色的起点。每个渐变点用百分比值表示，0%就是整个渐变的起点，而100%则是整个渐变的终点。下面这个例子把橙色和黄色的范围扩展到了中间：

```
background: linear-gradient(from top, red 0%, orange 20%, yellow 80%,
  violet 100%);
```

linear-gradient()函数的语法很好掌握。有个坏消息：为保证对Android浏览器和稍旧一点的Safari（Safari 7之前）浏览器的支持，还需要加上开发商前缀-webkit-。更糟糕的是，-webkit-linear-gradient()函数和真正的linear-gradient()函数有一些细微的差别。与linear-gradient()不同，-webkit-linear-gradient()不使用from来指定方向，而是自动赋值。

下面是一段兼容了稍旧一点浏览器的样式代码，添加了一个带开发商前缀的渐变：

```
.colorBlendBox {
  background: lightblue;
  background: -webkit-linear-gradient(top left, white, lightblue);
  background: linear-gradient(from top left, white, lightblue);
}
```

幸运的是，不需要添加其他的开发商前缀（比如-moz-和-o-），除非你想要支持更老版本的Firefox和Opera。

提示　在所有这些例子中，渐变都是通过background属性实现的。实际上，对background-image属性使用同样的渐变函数也能达到相同的目的。唯一的区别是使用background-image属性，可以创建背景图像作为后备。方法相信不说你也知道，即先用background-image属性为那些后进的浏览器指定一个类似的背景图片，然后再对background-image属性应用渐变函数。支持渐变的浏览器一般也很聪明，除非必要，否则它们不会下载后备图片，从而节省带宽。

2. 放射性渐变

要创建放射性渐变，使用radial-gradient()函数。我们要为这个函数提供一个圆心颜色和一个圆周颜色，圆周与元素边界接触。下面这个放射性渐变其圆心是白色，然后逐渐过渡到圆周的蓝色：

```
background: radial-gradient(circle, white, lightblue);
```

保险起见，还是要加上一条带-webkit-前缀的：

```
background: -webkit-radial-gradient(circle, yellow, green);
```

如果要将渐变拉伸成椭圆形的以更好地适配容器，可以将circle修改为ellipse。

和线性渐变一样，我们可以选择很多颜色。可以通过添加百分比来调节从一个颜色渐变到另一个颜色的快慢。下面这个例子中，渐变从黄色开始，缓慢地渐变到绿色，然后在靠近元素边缘的位置，迅速地渐变到蓝色、白色和黑色。

```
background: radial-gradient(circle, yellow 10%, green 70%, blue, white,
black);
```

我们也可以用百分比来设置渐变的中心位置。比如，如果想要渐变的中心点靠近元素的右上角，可以用下面这种放射性渐变：

```
background: radial-gradient(circle at 90% 5%, white, lightblue);
```

这对百分比告诉浏览器渐变开始的位置为离左边缘90%（这几乎要到右边缘了）和离上边缘5%。

注意　radial-gradient()函数的语法已经和它诞生时不一样了，用以指定渐变中心点位置的at关键字，是一个较新的特性。虽然在radial-gradient()函数中使用at是安全的，但不要试图在有开发商前缀的-webkit-radial-gradient()函数中使用它。

3. 循环渐变

CSS3还有两个可以创建绚丽渐变的函数：repeating-linear-gradient()和repeating-radial-gradient()。linear-gradient() 和radial-gradient()只将所设置的颜色渐变一次，而repeating-linear-gradient()和repeating-radial-gradient()函数会以相同的颜色顺序无尽地循环渐变，直到颜色条纹填满了元素。这种醉人的扎染效果会让你恍惚觉得仿佛回到了20世纪70

年代。

repeating-linear-gradient() 和 repeating-radial-gradient() 函数的语法基本上与linear-gradient() 和radial-gradient()一样。唯一的不同是，你需要确保限制了渐变的大小，以使其可以循环。

比如，下面这个循环渐变的效果与普通渐变完全一样，因为它没有限制大小，而是从中心开始，以黄色渐变到边缘的绿色：

```
background: repeating-radial-gradient(circle, yellow, green);
```

下面的渐变就不同了，它设置中心的颜色为黄色，但设置绿色在10%的位置就结束。然后，渐变循环，又以黄色开始。产生的效果就是混合了黄色到绿色过渡的条纹：

```
background: repeating-radial-gradient(circle, yellow, green 10%);
```

循环渐变中你可以设置任意多的颜色。关键在于要确保最后的颜色包含了一个百分比或像素值，这设置了颜色的位置，而不是放在元素的边缘。

除了使用百分比，还可以用像素值，像下面这样：

```
background: repeating-linear-gradient(to top, red, orange, white, yellow,
red 30px);
```

这个渐变创建了一个稍微不一样的效果。现在每个条纹都有一样的宽度（30像素），条纹的数量取决于元素内的空间大小。通过比较可以发现，前面的例子中总是有10个按比例划分的条纹，每一个填充了元素空间的10%（确切地说，是条纹宽度与空间最大半径的比例）。

6

提示 有两个关于循环渐变的警示。首先，只能用to关键字，而不能用from，因为一个循环渐变只能向一个方向填充。其次，如果想要渐变的每一个循环过渡得平滑一些，而不是一个生硬的颜色变化，就要确保颜色列表中的最后一个颜色和第一个颜色一样。

优雅的渐变

创建复杂的渐变是个精细活儿。要加快这个过程，你可能想要一个在线的渐变生成工具。方案很简单：在网页中调节控制器直到渐变达到满意的效果，然后工具会生成所需的代码（包括不同开发商前缀的所有代码，以防万一需要）。有两个不错的渐变生成工具，分别是 Ultimate CSS Gradient Generator（www.colorzilla.com/gradient-editor）和微软的 CSS Gradient Background Maker（http://tinyurl.com/ms-gradient）。

6.3 创建过渡效果

在CSS2.1炙手可热的时候，开发者们对伪类（参见附录A）这个新特性感到非常兴奋。仿佛一夜之间，大家用:hover和:focus伪类，就能实现原来需要JavaScript才能做出来的效果。比如，

要为按钮创建鼠标悬停效果，只要为:hover伪类应用一组新样式即可。当访客鼠标移动到按钮上面时，浏览器就会自动为按钮应用这些组新样式。

提示　如果你还不知道怎么创建按钮的鼠标悬停效果，千万要看一看*Creating a Webaite: The Missing Manual*这本书。另外，看看这篇文章也行：www.elated.com/articles/css-rollover-buttons/。

伪类创造的交互特性虽好，但已经有点过时了。主要问题是——太突然了。换句话说，如果使用了:hover伪类，鼠标放上去马上换样式，鼠标一离开马上就没有，太突然了。太突然了就显示得不自然了。想想在Flash应用或者其他桌面应用中，类似的效果要优雅得多。当我们把鼠标悬停到某个按钮上时，其颜色会切换、位置会改变，甚至会发光，整个过程会持续一小段时间，而且是以动画效果完成。

不少Web开发人员已经开始在自己的网页中添加类似的效果了。但这些效果一般都要借助第三方的JavaScript动画框架来完成。实际上，CSS3提供了一个更简单的方案，即新的过渡（transition）功能，可以从一组样式平滑地切换到另一组样式。

6.3.1　基本的颜色过渡

要理解什么是过渡，最好是看一个例子。图6-10展示了一个按钮的颜色过渡，使用了CSS3的过渡功能。

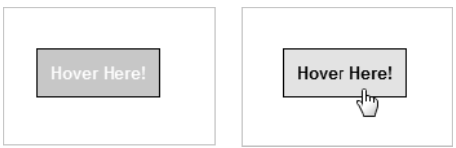


图6-10：如果这是普通的翻转按钮，那么鼠标一上来背景就会突然从绿色变成黄色。但使用了过渡之后，绿色会融入黄色，切换过程会持续半秒钟。鼠标离开按钮后，相同的过渡又会发生，只不过颜色切换的顺序相反，之后按钮恢复到初始状态。给人的感觉就是按钮很精致。（如果要试一下，可以访问http://prosetech.com/html5的试验页面）

首先，想一下如何以常规方式写这个按钮的样式，而不使用过渡。这是基本的CSS，需要用一条规则设置按钮的初始样式，再用一条规则设置鼠标悬停的样式：

```
.slickButton {
  color: white;
  font-weight: bold;
  padding: 10px;
  border: solid 1px black;
  background: lightgreen;
  cursor: pointer;
}

.slickButton:hover {
```

```
  color: black;
  background: yellow;
}
```

下面是一个使用这些样式规则的按钮：

```
<button class="slickButton">Hover Here!</a>
```

这种方式很好，但不够精致。为了更平滑地从绿变黄，需要用transition属性创建一个CSS3过渡效果。而相应的属性声明要写在正常状态下的.slickButton样式规则中（而不是:hover伪类规则中）。

最低限度也要为每个过渡设置两方面信息：要过渡的CSS属性和动画时长。在这个例子中，过渡的是背景颜色，而持续时间为0.5秒：

```
.slickButton {
  color: white;
  font-weight: bold;
  padding: 10px;
  border: solid 1px black;
  background: lightgreen;
  cursor: pointer;
  -webkit-transition: background 0.5s;
  transition: background 0.5s;
}

.slickButton:hover {
  color: black;
  background: yellow;
}
```

我想你一定注意到了，这个例子使用了两个过渡属性，而不是前面所说的一个。这是因为CSS3过渡的标准尚未定稿，并且一些浏览器仍然需要开发商前缀-webkit-。

亲手试验一下，会发现这个例子有个问题。这个悬停过渡按钮会切换两个样式：背景颜色和文本颜色。但是我们写的过渡属性只指定要过渡背景颜色。结果就是文本一下子会由白色变成黑色，而新的背景颜色则会慢半拍才出现。

有两个方法解决这个问题。第一个办法是用逗号作为分隔符，指定同时过渡背景和文本颜色：

```
.slickButton {
  ...
  -webkit-transition: background 0.5s, color 0.5s;
  transition: background 0.5s, color 0.5s;
  ...
}
```

另外，还有一个更简单的办法。如果你想过渡所有样式，而且希望所有过渡都同步完成，可以在原来指定属性名的地方指定all：

```
-webkit-transition: all 0.5s;
transition: all 0.5s;
```

目前，所有浏览器的最新版本都支持过渡。旧版的IE（IE9及其更早版本）还不支持过渡，加开发商前缀也不行。不过，即使浏览器不支持过渡好像也问题不大。因为就算忽略过渡属性，效果还是有的，只不过是突然切换，而不是平滑地过渡了。这还是不错的，因为这意味着网站可以添加过渡功能，但同时又可以为旧版本浏览器提供基本不变的样式。

6.3.2 更多的过渡思路

利用CSS过渡把简单的颜色过渡提升到精致的程度已经不错了。可是，如果你想做出更加好玩的翻转按钮或菜单，那么可以考虑更多可以过渡的属性。以下是一些还算不错的建议。

- **透明度**。通过修改`opacity`属性，可以实现图像的淡入淡出。只是要记住，别把图像变得完全透明，否则访客根本不知道图像在哪。
- **阴影**。前面6.2.4节已经介绍了`box-shadow`属性，通过它可以为任何盒子元素添加阴影。而且合适的阴影也可以制作出漂亮的悬停效果。特别是没有偏移但有模糊的阴影，可以用来生成经典的发光效果。当然，利用`inset`阴影也可以做出内发光来。
- **渐变**。把线性渐变改成放射性渐变——不管怎么说，这种效果想错过都难。
- **变形**。6.3.4节将会介绍，利用变形可以移动元素、调整元素大小，甚至可以对元素任意变形。这些效果当然也是过渡的首选。

另外，对内边距（`padding`）、外边距（`margin`）和字体大小（`font-size`）应用过渡不值得考虑。这些过渡操作会耗费更多电量（因为浏览器要重新计算布局大小或文本提示），而且可能导致响应迟顿和卡壳。如果你想移动、放大或缩小元素，那么最好还是使用变形技术（参见6.3.4节）。

6.3.3 用JavaScript触发过渡

如你所见，过渡会在元素从一个样式切换到另一个样式时生效。如果想用一个优雅的、节省代码的方式来实现，可以用像`:hover`和`:focus`这样的伪类。但这种方式有个明显的限制。比如，这种方式不能让过渡在另一个时间发生或者响应另一个事件，也不能让过渡由一个元素触发然后对另一个元素起作用。像这样的情况，需要搞点JavaScript代码帮忙。

幸运的是，创建一个JavaScript驱动的过渡并不难。和普通过渡一样，先要创建两个样式规则，一个是元素的初始状态，一个是过渡结束的状态。然后用JavaScript代码查找相应的元素并在合适的时机修改它的样式。

不要抛弃老版本浏览器

我们知道，不支持过渡的浏览器会突然切换样式，但这还是不错的。可是，如果你使用了装饰性的CSS3样式（比如带阴影或渐变的按钮），那么老版本的浏览器则会完全忽略这些样式。这样就不太好了。这意味着使用老浏览器的访客根本**看不到**鼠标悬停的效果。

解决这个问题的办法就是使用老浏览器支持的后备样式。比如，可以创建一个使用不同背景颜色的后备悬停规则，**然后**再为悬停规则设置渐变。这样老浏览器至少能在鼠标悬停时显示

背景变化。而支持渐变的浏览器则会切换到渐变填充状态。要想实现更精确的控制，可以考虑Modernizr，这样就能为老版本浏览器添加完全不同的样式（参见1.6.7节）。

图6-11展示了一个用代码驱动过渡的例子。在这个页面中，两张图片相互遮盖——同一城市天际线的日景图和夜景图。按钮用了几行简单的JavaScript代码来触发一个显示或隐藏夜景图的过渡。

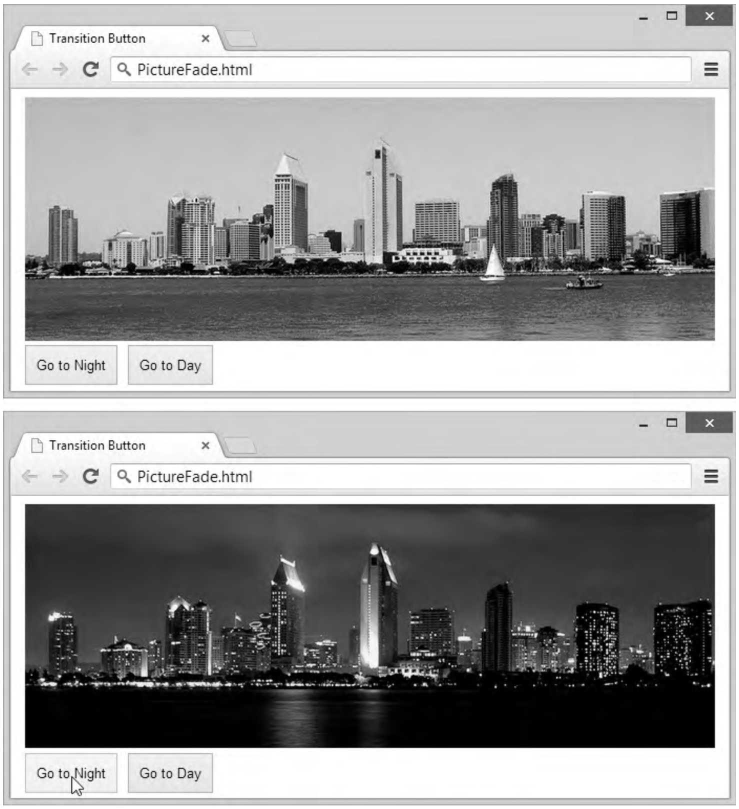


图6-11：开始时，夜景图是完全透明的（上）。点击To Night按钮后夜景图淡入，渐渐遮盖日景图（下）

创建这个例子的第一步是添加一个给图片格式化的样式规则，主要做两件事：设置图片为绝对定位（这样就能在包含它们的<div>元素中用一张图片覆盖另一张），然后定义要使用的过渡类型。这里用的是一个历时10秒的改变夜景图透明度的过渡。

```
img {
  position: absolute;
  -webkit-transition: opacity 10s;
  transition: opacity 10s;
}
```

我们还需要两条样式规则代表夜景图的两种不同的状态，开始时是完全透明的，但是在点击

按钮时可以变得不透明。

```
.solid {
  opacity: 1;
}

.transparent {
  opacity: 0;
}
```

HTML代码将两张图片都放在一个<div>内，把transparent类赋给第二张图片，还定义了两个按钮，并将点击事件回调函数写死在标签中。

```
<div>
  <img src="day.png" alt="Daytime view">
  <img id="nightImage" src="night.png" alt="Night-time view"
class="transparent">
</div>

<button onclick="toNight()">Go to Night</button>
<button onclick="toDay()">Go to Day</button>
```

下面是响应按钮点击的函数代码，找到夜景图，然后切换它的样式。

```
function toNight() {
  var nightImage = document.getElementById("nightImage");
  nightImage.className = "solid";
}

function toDay() {
  var nightImage = document.getElementById("nightImage");
  nightImage.className = "transparent";
}
```

虽然这看上去就像一个生硬的行为，但实际上是在10秒内平稳地变化，这得益于给所有<img>元素定义的过渡。

记住，只有当用户使用现代浏览器浏览时才有过渡。如果在IE9中浏览并点击To Night或者To Day按钮，页面会生硬地切换样式，没有10秒的过渡效果。不幸的是，还没有填补这个缺口的办法，并且当你开始使用过渡时，很容易忽视页面在旧版浏览器上会是什么样子。

注意 如果动画效果是你的页面所必需的一部分，可能你还没有完全准备好使用CSS3，毕竟目前使用最多的过渡解决方案是像jQuery UI或MooTools这样的JavaScript库。但毋庸置疑的是，当标准敲定并且现代浏览器遍布全世界的时候，CSS3一定是Web特效的未来。

6.3.4 变换

变换是一个强大的工具，可以移动、缩放、斜切和旋转元素，扭曲它的外观。利用CSS3变换，则可以改变元素的外观。与过渡类似，变换也是一个新的、试验性的功能。要使用变换，需

要用transform属性。下面是一个旋转元素及其内容的例子：

```
.rotatedElement {
  transform: rotate(45deg);
}
```

要让变换在Chome、Safari和Opera上生效，需要添加-webkit-前缀。在IE9中，需要添加-ms-前缀（而IE10及其更新版本不需要任何前缀）。Firefox不需要前缀。所以除非你在用-prefix-free库（参见6.2节），否则使用变换的正确方式应该是像下面这样：

```
.rotatedElement {
  -ms-transform: rotate(45deg);
  -webkit-transform: rotate(45deg);
  transform: rotate(45deg);
}
```

让过渡更自然

transition是一个组合了很多细节的集大成属性。到目前为止，你已经知道了如何给过渡指定时长和变化的样式属性。然而，还有两个可以用来调整过渡的细节。

首先，可以用一个调速函数来控制过渡的运行效果，比如，开始很慢然后加速或者开始很快然后减速。在时间短的过渡里，调速函数的效果不明显。但是在长一点的、更复杂的动画中，它能彻底改变效果。下面这个例子用了ease-in-out这个调速函数，这样过渡开始时慢，然后加速，然后慢下来直到最后：

```
transition: opacity 10s ease-in-out;
```

其他调速函数有linear（过渡的速率从开始到结束是恒定的）、ease-in（过渡开始慢，然后以恒定的速率运行）、ease-out（过渡以恒定的速率开始，但在后面会慢下来）、cubic-bezier（过渡效果根据所定义的贝塞尔曲线运行）。

不管用什么调速函数，整个过渡耗时相同——你指定的时长，不同之处在于过渡运行时如何加速或减速。要查看不同的调速函数，并感觉一下它们如何改变过渡的步调，可以访问http://css3.bradshawenterprises.com/transitions/，在这里可以看到用一连串滚动的方块动画演示的不同调速函数的效果。

过渡的另一个可以添加的细节是一个可选的延时，用以将过渡的开始推迟一段时间。下面是一个等待0.1秒的例子：

```
transition: opacity 10s ease-in-out 0.1s;
```

在前面的例子中，我们使用rotate()函数把元素围绕其中心点旋转了45度。不过，还有很多变换函数，可以使单独使用，也可以结合使用。比如，下面的例子就连续使用了三种变换效果：先把元素增大一半（使用scale变换），再向左移动10个像素（使用scaleX变换），然后又斜切了10°（使用skew变换）：

```
.rotatedElement {
  -ms-transform: scale(1.5) scaleX(10px) skew(10deg);
```

```
  -webkit-transform: scale(1.5) scaleX(10px) skew(10deg);
  transform: scale(1.5) scaleX(10px) skew(10deg);
}
```

斜切的意思就是扭曲变形。想象一下，有一个纸箱子躺在那儿，口朝着你。你把它的顶部用力推向一侧，而底部还固定不动（结果就变成了平行四边形）。

表6-2列出了所有可用的二维变换函数。要移除所有的变换效果，可将transform属性设置为none。

注意 变换不会影响页面中的其他元素，也不会影响布局。例如，通过变换放大某个元素，那么该元素会简单地覆盖相邻元素。

表6-2 变换函数

函　　数	描　　述
translateX(x)	在水平方向移动元素，正值向右移动，负值向左移动
translateY(y)	在垂直方向移动元素，正值向下移动，负值向上移动
translate(x, y)	在水平和垂直方向移动元素
scaleX(x)	在水平方向缩放元素。比1.0大的值为放大（2.0是两倍大），0到1.0之间的值为缩小（0.5是一半大）。使用负值会将元素绕y轴翻转，创建一个从右到左的镜像
scaleY(y)	在垂直方向缩放元素。比1.0大的值为放大，0到1.0之间的值为缩小。使用负值会将元素绕*x*轴翻转，创建一个从下到上的镜像
scale(x, y)	在水平和垂直方向缩放元素
rotate(angle)	围绕元素中心顺时针旋转。用负值则逆时针旋转。可以用CSS的transform-origin属性让元素围绕其他点旋转
skewX(angle)	在水平方向倾斜元素。上下边缘仍然水平，左右边缘倾斜
skewY(angle)	在垂直方向倾斜元素。左右边缘不倾斜，上下边缘倾斜
skew(x-angle, y-angle)	在水平和垂直方向倾斜
matrix(n1, n2, n3, n4, n5, n6)	用矩阵乘法来变换元素。这个用6个数字表示的矩阵，可以复制任何其他变换（或任何变换组合）。但是，你不太可能自己创建所需的矩阵，即便你是个数学爱好者，也很可能会用工具来生成你想要的矩阵

注意 当你已经玩腻了在二维空间移动元素，可以用3D变换在三位空间移动、旋转和弯曲元素。可以在http://tinyurl.com/3d-transitions找到一些可交互的3D变换的好例子。

如何移动起始点

一般来讲，变换是以元素的中心点为参照点的。可以通过在应用变换前用transform-origin属性来移动这个参照点。比如，下面这个例子介绍了如何围绕元素的左上角旋转：

```
.rotatedElement {
  -ms-transform-origin: 0% 0%;
  -webkit-transform-origin: 0%  0%;
  transform-origin: 0%  0%;

  -ms-transform: rotate(45deg);
  -webkit-transform: rotate(45deg);
  transform: rotate(45deg);
}
```

要围绕右上角旋转，可以用100% 0%这个值。甚至可以指定一个不在元素上的很远的参照点（比如50% 200%，这样参照点位置在元素水平方向的中间，和从上边缘向下两倍于元素高度的地方）。

默认情况下，transform-origin属性设置为50% 50%，这刚好是元素的中心点。

6.3.5 使用变换的过渡

变换和过渡可以说是天生一对。假设你想创建一个在线的数字影集，如图6-12所示。

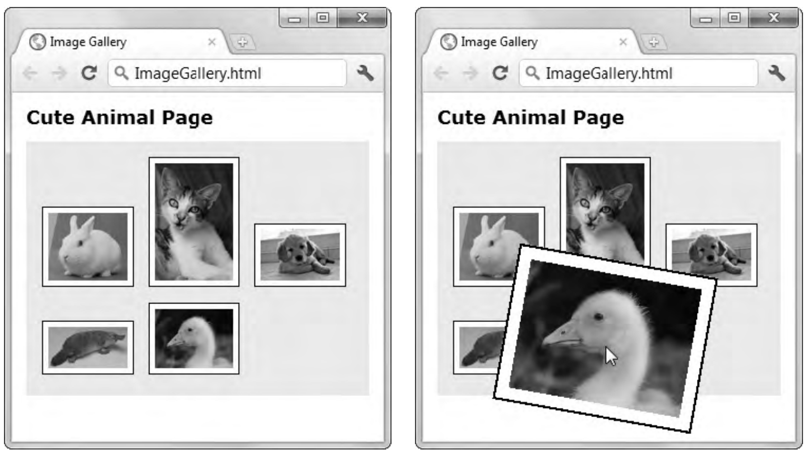


图6-12：在这个影集中，鼠标悬停后的变换效果让图片感觉跳了出来

这个例子的基本标记是很简单的，就是在一个<div>中放入一堆图片：

```
<div class="gallery">
  <img src="bunny.jpg">
  <img src="cat.jpg">
  <img src="dog.jpg">
  <img src="platypus.jpg">
  <img src="goose.jpg">
</div>
```

以下是为容纳图片的<div>元素应用的样式：

```
.gallery {
  margin: 0px 30px 0px 30px;
  background: #D8EEFE;
  padding: 10px;
}
```

下面则是为每个`<img>`元素应用的初始样式：

```
.gallery img {
  margin: 5px;
  padding: 5px;
  width: 75px;
  border: solid 1px black;
  background: white;
}
```

注意，这里为每个图片都设置相同的宽度（使用`width`属性）。这是因为根据例子的需要，必须在显示图片时，显示较大图片的缩小版。换句话说，这样浏览器在通过变换放大图片时才有余地。如果这里不是给大图片设置小一点的尺寸，而是统一显示缩略图大小的图片，那它们放大后就会模糊不清了。

接下来做悬停效果。当用户鼠标移动到图片上时，浏览器应该旋转并稍微放大一点图片：

```
.gallery img:hover {
  -ms-transform: scale(2.2) rotate(10deg);
  -webkit-transform: scale(2.2) rotate(10deg);
  transform: scale(2.2) rotate(10deg);
}
```

这样就可以通过变换把图片放大到新尺寸并旋转一定角度。而为了让这个效果看起来看自然流畅，还可以在正常状态下定义一个针对所有样式的过渡：

```
.gallery img {
  margin: 5px;
  padding: 5px;
  width: 75px;
  border: solid 1px black;
  -ms-transition: all 1s;
  -webkit-transition: all 1s;
  transition: all 1s;
  background: white;
}
```

好了，鼠标悬停时，图片的旋转和增大会在1秒钟内完成。而鼠标离开时，图片又会收缩并回到原位，同样还是用1秒钟时间。

6.4 Web 字体

在CSS3所有时髦的新功能中，很难说哪个最好。但是，假如非要找出那么一个功能，一个现在就能用，而且能够令人浮想联翩的功能，我想就要数Web字体了。

以前，Web设计人员只能使用少数几种安全字体。所谓安全字体，就是已知的所有浏览器和

操作系统都支持的字体。然而，任何一位有点经验的设计师都知道，字体在营造文档氛围的过程中，具有不可替代的重要作用。选择一款合适的字体，原本冷冰冰的学术说教，瞬间就会让人暇思无限，而从古典守旧到未来主义同样也只有一步之遥。

注意 为什么浏览器不急于实现自定义Web字体？首先，有一个优化的问题。由于计算机显示器的分辨率远远赶不上印刷的精度，所以如果Web字体设置不当，显示器在显示小号字体时就会模糊一团。其次，大多数字体并非免费。微软等大公司显然不愿意鼓励Web开发人员在未经允许的情况下，就把自己电脑里的字体上传到网站上，这可以理解。下一节我们马上会介绍到，字体公司对这两个问题都给出了解决方案。

CSS3通过`@font-face`为浏览器增加了强大的字体功能。使用这个功能的步骤如下：

(1) 把字体上传到网站（或者为了支持不同的浏览器，上传该字体的多个不同版本）；

(2) 使用`@font-face`命令注册每一个想要在样式表中使用的字体；

(3) 在样式表中使用注册过的字体，就像使用Web安全字体一样使用字体名字；

(4) 浏览器在遇到使用Web字体的样式表时，就会把字体下载到页面和图片的临时缓存中。然后就在你的网页或网站中使用该字体（如图6-13所示）。如果其他网页也要使用相同的字体，则需要分别注册并提供自己的字体文件。

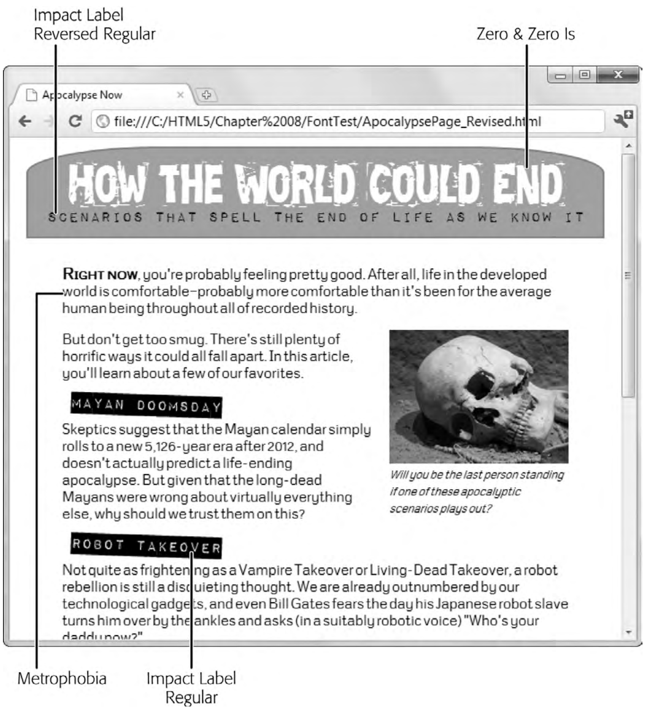


图6-13：修改后的启示录页面使用了4种新字体。这些字体都是免费的，6.4.3节将介绍怎么从Font Squirrel取得这些字体

> **注意** 严格来讲，`@font-face`不是新功能。CSS2当时就定义了这个命令，但由于浏览器开发商意见不统一，CSS2.1又把它给删除了。现在，CSS3又开始致力于把`@font-face`打造成一个普适的标准。

接下来几小节分别讨论上述几个步骤。

6.4.1 Web字体格式

尽管目前所有浏览器都支持`@font-face`，但它们支持的字体文件格式却不一样。Internet Explorer已经支持`@font-face`很多年了，但它只支持一种字体文件格式EOT（Embedded OpenType）。这种格式有很多长处，比如它支持通过压缩减少字体文件大小，也支持严格的网站许可，从而不会被其他网站盗用。可是，EOT格式一直没有发展起来，除了IE之外，其他浏览器都不支持它。直至最近，其他浏览器都还是支持桌面应用中常见的字体格式，即TTF（TrueType）和OTF（OpenType PostScript）。而除了上述几种格式外，还有另外两种字体格式：SVG和WOFF。表6-3列出了所有这些字体格式。

表6-3 嵌入字体格式

格　　式	说　　明	适用环境
WOFF（Web Open Font Format）	唯一一个面向未来的字体格式。比较新的浏览器支持它	IE9、Firefox 3.6和Chrome 6及更高版本支持它
EOT（Embedded OpenType）	微软特有的格式，除了IE没有别的浏览器支持	IE（IE9之前）
TTF（TureType）、OTF（OpenType PostScript）	桌面应用中常用的字体格式	使用Android操作系统的移动设备和（可选的）非IE浏览器，比如Firefox、Chrome、Safari和Opera
SVG（Scalable Vector Graphics）	一种用于字体的多功能图形格式，效果并不是太好（显示速度慢，而且文本质量不高）	旧版的移动版Safari（iOS 4.2之前）和（可选的）使用Android操作系统的移动设备

记住：要用`@font-face`功能并且支持大部分浏览器，需要将字体制作成多种格式。最佳实践方案是包含一个WOFF文件（在现代浏览器上性能最好）、一个EOT文件（兼容旧版IE）以及一个TTF或者OTF文件（兼容Android和旧版的非IE浏览器）。最好再为旧版的iPad和iPhone提供一个低质量的SVG文件。

如果觉得这样要维护太多的字体文件，可以缩减到两个文件并且可以支持大部分浏览器：一个TTF或者OTF格式的（任何一种都行）和一个EOT格式的。这不能兼容所有的浏览器，但可以为网站的绝大部分用户提供漂亮的字体。

> **注意** 幸运的是，字体开发商和在线字体服务通常会提供所需的全部4种格式，因此可以确保最大程度的浏览器支持。

6.4.2 给网站找个字体

现在我们知道从哪里可以获取网站所需的字体文件。要得到所需的格式，有两种可能：

- ❑ **下载一种免费网络字体**。采用这种方式时，不用担心许可问题，不用破费了。
- ❑ **将已有的桌面字体转化为网络字体**。这种方式可以让我们使用钟爱的字体。这也能实现一致性。比如，如果公司已经有一套字体标准，用于logo、备忘录和出版物，那么在线上用相同的排版样式就很合理 。但是，可能需要研究一下如何解决许可问题，可能还需要支付一点费用。

接下来几节我们会介绍这两种方式。

消除异常

即使你规规矩矩地提供了必要的字体格式，仍然免不了会遭遇一些异常情况。下面给出了在使用Web字体时偶尔会出现的问题。

- ❑ 很多字体在古老但依然有很多用户的Windows XP中看起来并不舒服，因为Windows XP通常会禁用反锯齿功能。（没有打开反锯齿功能时的字体看起来像是“乍毛鸡”。）
- ❑ 有人反映某些浏览器（或操作系统）在显示某些嵌入字体时有问题。
- ❑ 在有些浏览器中会出现所谓的FOUT（即Flash of Unstyled Text）问题。在需要几秒钟下载某种嵌入字体的情况下，页面会首先使用备用字体呈现文本，然后再使用嵌入字体重新渲染。老版本的Firefox中最常见这个问题。如果这对你而言已经成了必须解决的问题，可以使用谷歌提供的一个JavaScript库，利用它可以自定义未加载完成时使用的样式，完全控制渲染过程（参见http://tinyurl.com/font-loader）。

虽然在使用Web字体时偶尔会遇到这些小问题，但新版本的浏览器已经逐步把它们都解决了。比如，Firefox为了消除FOUT问题，会在使用备用字体之前，先等待3秒钟以下载嵌入字体。

6.4.3 从Font Squirrel获取免费字体

获取免费字体最好的地方就是Font Squirrel网站www.fontsquirrel.com，它提供了大概1000种免费使用的字体（见图6-14）。

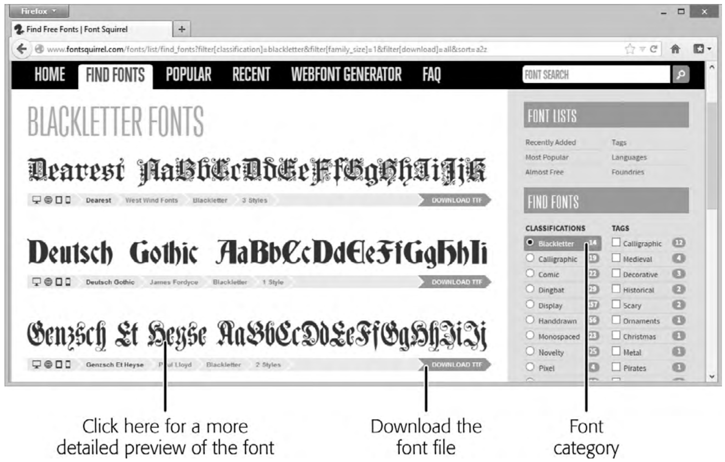


图6-14：Font Squirrel提供了多种搜索字体的方式，但最有效的方式还是直接输入想要的字体名称（比如Calligraphic、Novelty、Retro）。最棒的是，这里的大部分字体你都可以免费用在任何你需要的地方——在个人电脑上创建文档，或者在Web上构建一个网页

在Font Squirrel找到喜欢的字体后，查看一下下面的小图标（见图6-15），这些图标指明了字体的许可情况。实心图标表示该字体可以在特定范围内使用，空心图标则表示不能。



图6-15：前两个图标是最重要的，它们表示该字体可以用于桌面和Web。事实上Font Squirrel上所有的字体都包含这两个图标。接下来两个图标表示字体是否可以用于电子书和自定义应用程序。（如果不确定图标是什么意思，可以点击看一下）

核实了字体的许可细节之后（在Font Squirrel上都是这样），下一步就是进一步查看字体。点击字体文本切换到字体预览页面，这里显示了字体的每个字母，并可以让你输入一些文本来测试。

如果喜欢这个字体的话，最后一步就是下载了。你可能会下载到一个包含了Web字体的完整字体包，但更有可能的是下载到一个TTF或OTF文件，然后自己制作字体包，这取决于所下载的字体。这个问题的出现很大程度上是因为字体许可的混乱。很多字体使用SIL OPEN Font License许可，它允许用户免费使用字体，但不允许像Font Squirrel这样的服务商将字体重新打包。幸运的是，自己创建字体包很简单，并且完全合法。下一节我们会介绍怎么做。

在本地计算机中使用字体

我能不能在打印文档时也使用网页中的字体？

如果你在自己的网站中找到并使用了一款非常棒的字体，也可以在本地计算机中使用它。例如，可以用这种字体在Illustrator中创建一个logo。或者，你们公司想把它用在广告、产品手

册或财务报告中，打印出来。

目前的Windows和Mac计算机都支持TrueType（.ttf）和OpenType（.otf）字体。我们下载到的每个字体包中，通常都会包含其中一种格式——大多数时候是TrueTyep格式。要在Winddows中安装该字体，先把它从压缩文件中解压出来，然后在字体文件上单击右键，选择安装即可。（可以一次选择安装多个字体文件。）在Mac中，双击字体文件打开Font Book工具，然后再单击Install Font（安装字体）按钮。

6.4.4 准备一个网络字体

使用Font Squirrel，可以将标准的桌面TTF或OTF字体文件转化为可用于任何网页的Web字体。任何从Font Squirrel上下载的免费字体都可以这样做。你也可以用自己电脑上的字体来做，但重点是要理解许可问题，这是你首先要面对的问题（参见下文附注）。未经允许在网站上使用普通的桌面字体是可能侵犯版权的——如果字体厂商要求Font Squirrel将其字体列入黑名单，你就再也不能使用桌面到Web的字体转化了。

理解字体许可的规则

普通的用于桌面软件的字体不是免费的。将你电脑上的字体用于网站不太合适，除非你获得了字体开发者的明确许可。

比如，微软和苹果会为他们的操作系统和应用软件中用到的字体支付费用，这样你才能使用它们，比如，用文字处理软件写一篇新闻通讯稿。但是，这种许可并没有允许你把这些字体放到网络服务器上并在网页中使用它们。

如果你有一个喜欢的字体并想知道是否需要付费使用，唯一的办法就是联系开发该字体的公司或个人。一些字体开发者会基于网站的访问量收取许可费，其他一些开发者会收取极少的费用或不收费，但需要你满足一些条件（比如，在网站中包含一段关于所用字体的小字说明，或者你的网站是非营利性的）。还有一个附带好处：技术精湛的字体开发者经常提供针对Web展现优化过的字体。

获得某个字体的使用许可后，就可以使用Font Squirrel方便的Web字体生成器来转化它。点击Font Squirrel上方的Webfont Generator标签就可以访问生成器，或者直接访问www.fontsquirrel.com/fontface/generator，图6-16展示了操作步骤。



图6-16：首先，点击Add Fonts从电脑上传一个字体文件（第1步），勾选“Yes, the fonts I'm uploading are legally eligible for web embedding”，表示你接受他们的许可要求，如前文所介绍的那样（第2步）。最后，点击Download Your Kit（第3步）

下载字体包之后，得到的是一个包含很多文件的压缩文件。例如，如果下载Chantelli Antiqua字体，解压后可以得到如下文件：

```
Bernd Montag License.txt
Chantelli_Antiqua-webfont.eot
Chantelli_Antiqua-webfont.svg
Chantelli_Antiqua-webfont.ttf
Chantelli_Antiqua-webfont.woff
demo.html
stylesheet.css
```

其中的文本文件（Bernd Montag License.txt）包含授权许可信息，大意是你可以免费使用，但不能销售该字体。而4个Chantelli_Antiqua-webfont文件就是4种不同的文件格式。（根据选择的字体不同，还可能包含针对不同字形，比如粗体、斜体和纯黑体等的文件。）最后，stylesheet.css中包含着将该字体应用到网页的样式表规则，而demo.html则是一个显示该字体的示例页面。

要使用Chantelli Antiqua字体，首先要把所有Chantelli_Antiqua-webfont文件复制到网页所在的文件夹中。然后，就是注册该字体，以便在样式表使用它。为此，需要在样式表的开头写一个复杂的@font-face规则，如下所示（带着行号便于下文说明）：

```
1   @font-face {
2     font-family: 'ChantelliAntiquaRegular';
3     src: url('Chantelli_Antiqua-webfont.eot');
4     src: local('Chantelli Antiqua'),
5       url('Chantelli_Antiqua-webfont.woff') format('woff'),
6       url('Chantelli_Antiqua-webfont.ttf') format('truetype'),
7       url('Chantelli_Antiqua-webfont.svg') format('svg');
8   }
```

为理解这条规则都做了什么，下面我们逐行解释。

- 第1行：@font-face是正式注册字体的工具，注册之后才能在样式表的其他地方使用该字体。
- 第2行：给这款字体起个名字，具体叫什么取决于你；将来使用该字体时要用到这个名字。
- 第3行：必须首先注册EOT格式的字体文件，这样IE不理解其他规则也没有关系，只要忽略其他格式就行了。这里的url()函数用于告诉浏览器在当前位置下载另一个文件。如果把字体放在了网页所在的文件夹中，那么在此只要给出字体文件名即可。
- 第4行：接下来就要使用local()函数，这个函数告诉浏览器这种字体的名字，如果恰好访客的电脑中安装了这种字体，浏览器就会使用它。不过，个别情况下这个函数也会导致问题（比如，在Mac OS X中，根据访客字体的安装位置不同，可能显示一个安全对话框；另外，也可能会加载另一个只是同名的字体。）为此，Web设计人员有时候会在此传入一个明显瞎编的字体名，让浏览器在本地找不到该字体。比如，有人会使用一个没有意义的笑脸符号，如local('☺')。
- 第5行～第7行：最后一步是告诉浏览器可以使用的其他文件格式。如果字体包里有WOFF字体文件，建议把它放在第一位，因为这种格式的字体质量最高。然后，再注册TTF或OTF文件，最后注册SVG文件。

提示　当然，你完全可以不用自己动手写@font-face规则（也不必理解上面介绍的所有技术细节）。只要把Web字体包中包含的stylesheet.css文件中的规则复制到页面中即可。

使用@font-face注册了字体之后，就可以在样式表使用该字体了。这时，只要使用font-family属性，然后指定之前用@font-face给字体起的名字（第2行）即可。下面是一个例子，删除了@font-face规则的大部分代码：

```
@font-face {
  font-family: 'ChantelliAntiquaRegular';
  ...
}

body {
  font-family: 'ChantelliAntiquaRegular';
}
```

这条规则为整个网页应用了新注册的字体。当然，只通过元素或类名来缩小应用字体的范围也没有问题。唯一要注意的是，必须在使用字体之前，先使用@font-face注册字体。把上面的步骤改成先使用后注册，将导致无法使用字体。

6.4.5　用谷歌更简单地使用Web字体

如果你想使用新奇的字体，而又不想像前面介绍的那么麻烦，还可以使用谷歌提供的Web字体解决方案。这个项目名叫Google Fonts（原来叫Google Web Fonts），其中包含很多可以自由使

用的字体。使用Google Fonts，不必担心字体格式，因为谷歌会检测用户的浏览器并自动发送正确的字体文件。你只需要添加一个谷歌生成的样式文件。

要在页面中使用谷歌的Web字体，大致要遵循下列步骤。

(1) 在浏览器中打开www.google.com/fonts。

谷歌会列出很多字体来（图6-17）。



图6-17：谷歌一直坚持不懈地扩充这个字体列表。在查找字体时，可能要先选择排序方式，指定筛选选项（在圆圈位置）。例如，可以按照字母顺序排序，或者把最受欢迎的字体排在前面；而筛选选项包括只显示衬线（serif）、非衬线（sans-serif）或手写（handwritten）字体

(2) 在页面顶部，单击一个选项卡（Word、Sentence或Paragraph），选择预览字体的方式。

如果你想找一款字体用在标题上，那么可以选择Word或Sentence，看看一个单词或一行字的效果。如果你想找一款正文字体，那么就该选择Paragraph，体验一下整段文本的效果。无论选择哪种预览方式，都可以自己手工输入一些文本，并设置字体大小。

(3) 设定搜索选项。

如果你知道自己要找什么字体，可以在搜索框中输入该字体的名字。否则，就要不断滚动页面，多花点时间。为了节省时间，最好先选择一种排序方式，并添加一些筛选设置（比如，要想只显示最受欢迎的粗体非衬线字体，可以将排序方式设定为Popularity，在筛选设置中只选择Sans-Serif，而在Thickness中设置较粗的笔画）。图6-17展示了设置这些选项的地方。

(4) 找到一款满意的字体后，单击Pop out。

谷歌会弹出一个新窗口，其中包含该字体的详细说明及每个字符的展示。

(5) 如果你喜欢该字体，单击Quick-use取得使用它的信息。

然后，谷歌会给出使用该字体的代码。包括一个样式表链接（必须放到你的网页中）和一个使用该字体的样式规则示例。

(6) 把样式表链接添加到网页中。

比如，你选择了Metrophobic字体，那么应该把谷歌给出的以下链接放在<head>部分中：

```
<link href="http://fonts.googleapis.com/css?family=Metrophobic" rel="stylesheet">
```

这个样式表用于注册字体，使用了@font-face，因此你就不用自己注册了。最重要的是，谷歌会负责提供相应的字体文件，而你就不必再往自己的网站中上传任何东西了。

注意 别忘了把谷歌字体样式表的链接放到其他样式表链接的前头。这样，其他样式就可以使用谷歌的Web字体。

---

(7) 使用选择的字体（指定字体名）。

例如，要在标题中使用刚刚注册的Metrophobic字体，同时再添加几个后备，以防浏览器下载字体文件失败，可以像下面这样写样式规则：

```
h1 {
  font-family: 'Metrophobic', arial, serif;
}
```

创建字体集合

以上步骤是取得字体标记的最简便方式。不过，通过创建**字体集合**，还可以选择更多字体。

字体集合就是一种打包注册多种字体的方式。要创建字体集合，只要单击每款字体旁边的Add to Collection 按钮即可。每添加一种字体，该字体名就会出现在页面底部的蓝色背景之上。

选择完所有字体后，单击页面底部的 Use 按钮，就可以看一个与单击 Quick-use 之后看到的类似的页面，只不过此时页面中显示的是支持字体集合中所有字体的样式表链接。

创建了字体集合后，还可以使用页面右上角的两个按钮。单击 Bookmark your Collection（看起来像一截链条的那个）可以在浏览器中创建一个书签，以便将来对字体集合进行调整。单击 Download your collection（看起来像一个向下的箭头）可以把字体下载到本地计算机中，随后可以安装这些字体并用于文档打印。

提示 还在找完美字体？流行字体订阅网站，如http://fonts.com和http://typekit.com，提供了成千上万种超高品质的字体，它们来自一些非凡的字体开发商，比如Linotype和Monotype。对字体痴迷的Web开发者们会为此每年支付10到100美元，要在访问量巨大的超级网站上使用，费用还会更多。

6.4.6 多栏文本

使用自定义字体并不是CSS3在文本显示方面唯一的创新。多栏文本也是CSS3的一个模块，用于灵活地显示长篇内容，提高可读性。

要实现多栏文本，可谓易如反掌。而且，有两种方式可供选择。第一个选择是使用column-count属性来设置想要的栏数，比如：

```
.Content {
  text-align: justify;
  column-count: 3;
}
```

在本书写作时，只有IE10和IE11支持column-count属性。虽然Chrome、Firefox、Safari和Opera也支持多栏文本，但需要使用带开发商前缀的属性，如下所示：

```
.Content {
  text-align: justify;
  -moz-column-count: 3;
  -webkit-column-count: 3;
  column-count: 3;
}
```

这种创建多栏的方式适合固定布局。如果网页区域会随着浏览器窗口缩放而变化，那么这些栏可能会变得过宽，让人们无法阅读。此时，最好的方法是不要设置栏数，而是使用column-width属性告诉浏览器每一栏有多宽：

```
.Content {
  text-align: justify;
  -moz-column-width: 10em;
  -webkit-column-width: 10em;
  column-width: 10em;
}
```

这样，浏览器就会根据需要创建栏，以填充有效的空间（参见图6-18）。

注意　虽然在指定栏宽时可以使用像素单位，但使用em单位才是首选。因为em单位与当前字体大小是匹配的，所以如果网页访客调大了浏览器中的字号，那么栏宽也会按比例加大。具体来说，1em等于两倍字体大小，因此对于12像素的字体，1em就是24像素。



图6-18：在窗口很窄的情况下（上），Firefox会只显示一栏。随着窗口变宽，栏数也会相应增加（下）

CSS3还提供了一些用于装饰分栏样式的属性。可以用column-gap调整分栏之间的间隔。也可以用column-rule来添加一条垂直的分隔线，这个属性值的三项分别代表线条的宽度、样式和颜色（就像border属性）。下面这个例子添加一条红色的1像素宽的分隔线：

```
-webkit-column-rule: 1px solid red;
-moz-column-rule: 1px solid red;
column-rule: 1px solid red;
```

也可以用column-span属性让图片或其他元素横跨多列。column-span默认值是1，意味着这个元素锁定在它出现的那一列。其他的值只有all，让元素横跨所有列的总宽度。目前还没有办法让元素横跨指定数量的列（1列和全部列除外）。

下面这个例子（见图6-19）让一张图片横跨所有列：

```
.SpanFigure {
  -moz-column-span: all;
  -webkit-column-span: all;
  column-span: all;
}
```



图6-19：这个多列页面在列之间用了一个分隔线，并让一张图片横跨所有列

如果图片样式的`float`属性为非`none`的值，那么这种方法将无效。这是因为浮动的图片已经具备浮动于布局和它所包含的任何列之外的能力。

注意 如果需要在宽布局上打散文本以方便阅读，多列就很合适。但是，如果内容非常多的话，多列就不是最佳选择了，因为目前还没有办法让列的高度适配浏览器窗口的高度。因此，如果将一篇冗长的文章分成3列，读者就需要从顶部滚动到底部读第一列，然后回到顶部，再滚动到底部读完第二列，然后读第三列。如果内容超过两屏，还是用传统的一列式滚动吧。

第 7 章

CSS3与响应式Web设计

网页设计师要把内容放到HTML页面中，首先会面临一个挑战。平面设计师都会假定将来的文档如何在纸上排版，读者会怎么阅读，但网页就宽松多了，而且没有一定之规。用户的浏览器（以及个性化设置）决定了同一个HTML页面，是被挤到一个小小的窗口里，还是孤零零地飘浮在一个巨大的屏幕上。这就是网页设计师要面临的布局风险。在一个窗口里看起来几近完美的布局，到了另一个比例不同的窗口里，马上显得粗制滥造，丑陋不堪。

今天，这种不确定的风险有增无减。网页设计师不光要考虑桌面计算机中浏览器窗口的不同大小，还得适应平板电脑、智能手机等设备的不同大小。这样一来，网站布局的复杂度就会直线上升，特别是在大多数网站都会包含菜单、导航、边栏等部件的情况下。如果你的目标就是只创建一个网站，让它能在不同的浏览器环境下平滑过渡，那网页中的这些部件就构成了非常大的挑战。

因为网页设计师一直都依靠CSS来实现页面布局和格式化，所以最好让CSS针对上述问题拿出一个解决方案。CSS3确实为此提供了一个完美的工具：媒体查询（media query）。通过媒体查询，可以让网站根据窗口大小或设备的不同，无缝地在不同的样式集之间切换。

媒体查询是移动Web开发的基本技术。就算你不准备让访客通过手机浏览你的网站，媒体查询也还是有用的，它可以确保网站能够自适应浏览器窗口大小的变化。比如，窗口缩小时隐藏布局中的一栏，或者把导航链接从页面顶部转移到页面一侧。这种自适应可以归结为现在广为流行的一个设计理念：响应式设计，而这正是本章所要探讨的话题。

7.1 响应式设计基础

窗口大小不定是Web诞生之初就存在的问题。经过多年的实战，网页设计师已经为此发明了很多技术，有的精妙，有的粗鄙，但目标都是为了实现响应式设计。

在学习使用媒体查询之前，有必要先了解实现响应式设计的传统技术。这些技术即使到了今天也很重要，只不过正如我们将要说明的，面对今天的挑战，光有这些技术已经不够了。只有认识到它们的局限性，才能真正理解CSS3怎么填补这项空白。

7.1.1 流式布局

应对窗口大小缩放的最简单方案，就是做一个等比缩放布局。这种布局会尽其所能地占用所

有可用空间，不管空间多大或多小。

创建等比缩放布局在理论上相当简单，就是将网页分成几栏，然后用比例而非像素设定它们的宽度。假设我们有下面这样的两栏布局：

```
<body>
  <div class="leftColumn">
    ...
  </div>

  <div class="rightColumn">
    ...
  </div>
</body>
```

如果是固定布局，那么CSS规则是这样的：

```
.leftColumn {
  width: 275px;
  float: left;
}

.rightColumn {
  width: 685px;
  float: left;
}

body {
  margin: 0px;
}
```

如果是等比缩放布局，那么CSS规则就要这样写：

```
.leftColumn {
  width: 28.6%;
  float: left;
}

.rightColumn {
  width: 71.4%;
  float: left;
}

body {
  margin: 0px;
}
```

这里把左栏的宽度设定为28.6%，也就是占用容器宽度的28.6%，容器就是`<body>`元素。我们这个例子里的`<body>`元素没有外边距，因此与浏览器窗口是等宽的，那么左栏最终的宽度相当于浏览器窗口的28.6%。

结果，左右两栏的宽度加起来正好是100%，不管浏览器窗口多大或多小，这两栏都会随着扩张或收缩。等比缩放布局又称为*流式布局*，因为布局中的内容始终会像水一样，平滑地充满所有空间。

注意 在这个例子中，左栏28.6%的宽度是用它在固定布局中的宽度（275像素）除以固定布局总宽度（960像素）计算得来的。实际开发中，大家都习惯开始的时候先使用固定布局，然后在此基础上创建流式布局。

当然，光调整栏的大小还不行。还得考虑外边距、内边距和边框。新手网页设计师在第一次创建流式布局时，经常会设定固定的外边距和内边距（使用像素值），只等比缩放栏宽。想想看，栏所占的宽度必须减掉外边距宽度。可栏宽百分比是基于整个页面宽度计算的，并没有考虑被外边距占掉的空间。这种冲突在窄窗口中可能会导致问题，造成固定宽度的外边距把等比缩放的栏挤出第一排的现象。

举个例子吧，假设有以下样式表：

```
.leftColumn {
  width: 27%;
  margin: 5px;
  float: left;
}

.rightColumn {
  width: 68%;
  margin: 5px;
  float: left;
}
```

这里两栏合计占95%，给外边距留出5%的空间。在中大尺寸窗口中，5%足以放得下外边距了，但如果窗口窄到一定程度，就会导致5%的空间容纳不下固定的外边距。要想看看出问题时的效果，可以用background属性给每一栏设定不同的背景颜色，然后逐渐调窄窗口，如图7-1所示。

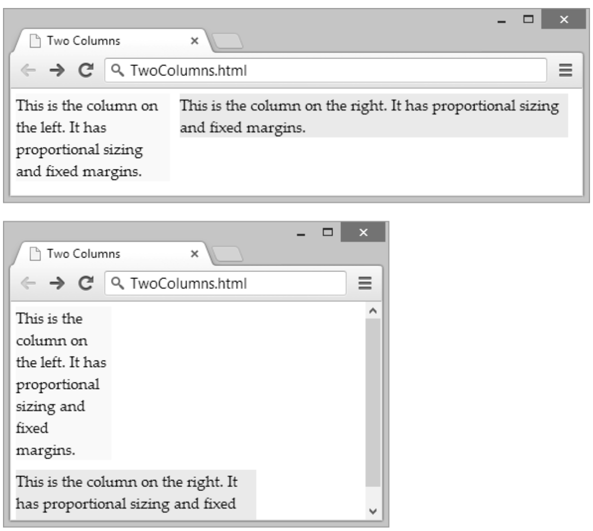


图7-1：等比缩放的栏与固定宽度外边距不适配。在窗口过小的情况下，外边距会把第二栏挤到下面

要解决这个问题，必须同样以比例设定外边距的宽度。如果除了栏页面还剩下5%的宽度，那么就可以把这个宽度分配给外边距。平均分三份，每份是1.66%，分别位于窗口左边、窗口右边和两栏之间，这样：

```
.leftColumn {
  width: 27%;
  margin-left: 1.66%;
  margin-right: 1.66%;
  background: #FFFFCC;
  float: left;
}

.rightColumn {
  width: 68%;
  margin-right: 1.66%;
  background: #CCFFCC;
  float: left;
}
```

图7-2展示了这样修改后的结果，此时外边距和栏宽都会按照比例缩放。

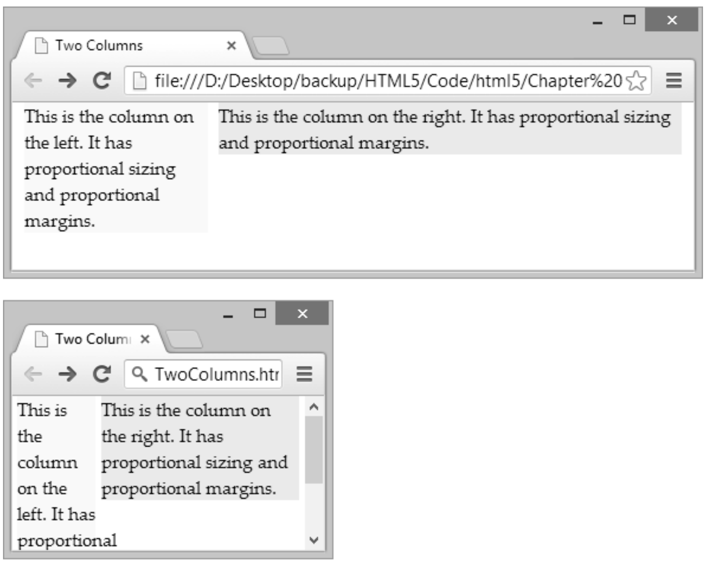


图7-2：在水平方向上通过百分比设定所有宽度之后，页面布局就可以整体缩放，适应任意窗口大小了

有时候，外边距基于浏览器窗口等比缩放可能不是我们想要的效果。为此，可以想一个别的办法。比如，在等比缩放的其中一栏里再额外增加一个元素，给这个元素设定固定宽度的外边距和内边距。由于这个元素位于等比缩放的顶级部局元素（栏）的内部，因此它的宽度会自动适应窗口大小变化。

边框也有类似的问题。如果你给栏加了边框，那么因为它也要占用空间，所以也会像图7-1中的固定外边距一样破坏布局。但边框宽度不接受百分比值，等比缩放边框的方法不适用。此时

最简单的办法，同样是在等比缩放的栏里额外增加一个<div>元素，把边框设定到这个<div>元素上。这个技巧已经被使用很多年了，虽然会增加一点标记的复杂性（多了一层布局元素），但它可以保证布局在任意尺寸的窗口下都不走样。

CSS3的box-sizing与calc()

刚才我们谈到的布局问题是一个常见的问题，常见到CSS3为此给出了很多的解决方案。在此我们只介绍两个最有前途（但也不能算完美）的方案。

- box-sizing。通常，边框会被加在元素的外围，因此在计算布局时就要减掉边框占用的空间。CSS3增加了一个box-sizing属性，如果这个属性的值是border-box，那么边框就会位于盒子的内侧。边框看起来没什么区别，但盒子大小的计算方式不一样了。比如，这意味着67%宽的栏，无论其边框多粗，始终都是67%宽。
- calc()函数。如果你需要混合使用比例和固定单位，可以让CSS3帮你计算一个结果，然后在布局中，这就要用到calc()函数。举个例子，假如你想把一栏的宽度设定为67%减5像素（的外边距）。不太细心的开发者可能会直接把栏宽设定为65%（这样会导致图7-1所示的空间不适配的问题）。但有了CSS3，就可以把width属性设定为calc(67%-5px)，以确保这一栏始终能占据除外边距之外的所有空间，一个像素也不多。

可惜的是，使用这两个新特性的麻烦也不少。IE7不支持box-sizing属性，在Firefox中使用它需要添加开发商前缀-moz-（参见6.1.4节）。IE7、IE8都不支持calc()函数，在较早的Android Chrome和Safari浏览器中使用它也需要添加-webkit-前缀。当然也有一些腻子脚本可以帮我们“抹平”这些问题，但目前来看，在用户升级到较新的浏览器之前，最好还是先不使用它们。

7.1.2 流式图片

要实现响应式设计，构建多栏等比缩放的布局是第一步。第二步就是处理栏中的内容，这一步涉及的问题更多。

其中一个问题就是图片。一般来说，图片占用的空间取决于其内容，也就是图片到底由多少个像素组成。但这种不加控制的处理方式在小尺寸窗口中会导致问题。如果窗口太小，图片就会撑开所在的栏，挤掉其他元素，让人觉得布局过于草率。

解决这个问题的方法很简单，就是把图片的宽度限定为其容器的最大宽度：

```
img {
  max-width: 100%;
}
```

一如既往，这里的100%相对的是元素所在的容器。在此，容器就是栏，而不是整个页面。结果，图片要么显示为自己的实际大小，要么扩张到容器的边界为止，如图7-3所示。

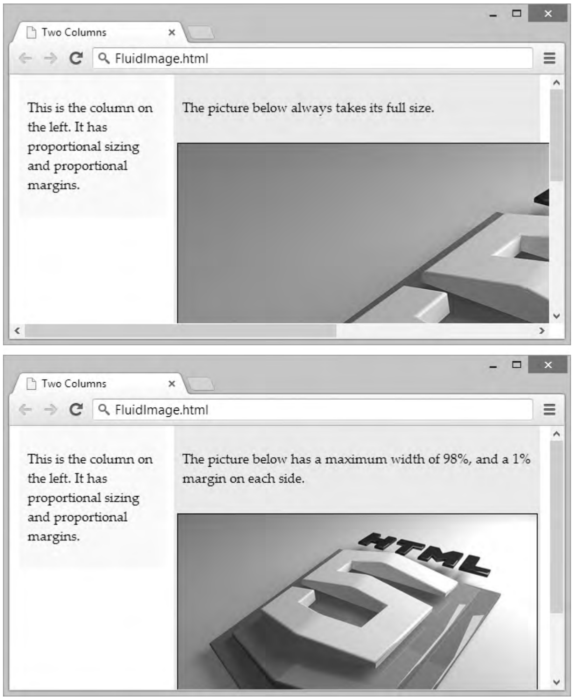


图7-3：没有设定流式图片与设定流式图片的前后对比，前者图片窜出了窗口，后者图片以适当尺寸显示

提示 如果你想给图片周围加上一点外边距，必须给margin-left和margin-right设定百分比值，而且要保证max-width等于100%（不能超）。

流式图片这个方法也有不足，那就是无论显示的时候多小，浏览器都必须下载完整的大图片。这样就会浪费一些时间和带宽，移动设备对此比较敏感。可惜CSS本身没有办法解决这个问题！别急，有人想出了解决方案，但需要用到服务器端代码、Web服务和JavaScript库。如果你的网站要向移动用户显示大量图片，可以考虑这种方案，参见*Smashing Magazine*的这篇文章：http://tinyurl.com/responsive-img（不过，大多数Web开发者不用考虑这些）。好在，对于类似但更严重的视频缩放问题，我们还是有原生方案的，参见本章最后的附注栏。

7.1.3 流式排版

知道了怎么实现流式布局和流式图片，接下来该关心栏中的文字内容了。粗心的Web开发者会给文字选择差不多大的字号（使用像素单位），然后就那样了。可是，这种“硬编码”的字号会破坏响应式布局，因为在大显示器上看着舒服的文字，到了移动设备的小屏幕上就会变得难以辨认。为看清文字，访客必须手工放大网页。响应式布局应该做到不需要访客缩放或滚动页面，就能适应各种屏幕大小。

同样，解决方案还是不要使用像素或点这样的固定度量单位，而要使用百分比或em等相对单位。所谓em，就是M的读音，是最常用的相对单位。

注意 在传统铅字排版时代，em是表示字母宽度的单位。那时候说的em dash（全身线），指就是与今天字体中大写M等宽的线。[1]

百分比和em的结果相同，都是让文字相对于浏览器默认的文字大小缩放。把文字大小设置为110%或1.1 em，结果就是比常规没有应用样式的文字大10%，而50%或0.5 em就是常规文字大小的一半。

虽然使用百分比和em都不是问题，但大多数响应式布局的开发者都沿习同样的做法。他们把页面的基准文字设置为100%（只是为强调这是其他文字缩放的基准），然后在其他元素中再用em单位放大或缩小文字：

```
body {
  font-size: 100%
}

p {
  font-size: 0.9em
}

h1 {
  font-size: 2em
}
```

富有经验的Web开发者不会就此罢手。他们会把布局中所有固定大小的度量单位一律改成em单位。比如说，布局的某个角落有一个边框或者一点外边距或内边距，最好也用em而不是像素。使用em之后，这些细节所占用的空间都会根据文字大小而缩放。变化虽然不大，但却能给人更专业的印象。

下面举个具体的例子。假设左边的一栏是两个<div>嵌套的布局，内部的<div>用于为左栏内容周围添加空白，这样不会影响整体两栏布局的等比缩放：

```
<body>
  <div class="leftColumn">
    <div class="leftColumnContent">
      ...
    </div>
  </div>

  <div class="rightColumn">
    ...
  </div>
</body>
```

注1：参见译者的文章“你未必知道的CSS故事：揭开leading的面纱”：http://ituring.cn/article/18076。（译者注）

用像素来设定左栏中的内容<div>当然也行，布局也是流式的。但使用em的效果会更棒，就像下面这样：

```
.leftColumn {
  width: 28.6%;
  background: #FFFFCC;
  float: left;
}

.rightColumn {
  width: 71.4%;
  background: #CCFFCC;
  float: left;
}

.leftColumnContent {
  border: 0.07em solid gray;
  margin: 0.3em;
  padding: 0.2em 0.3em 0.4em 0.4em;
}
```

注意 对大多数布局而言，使用em单位设定边框、内边距和外边距，最大的好处是可以防止元素在小窗口里显示得过大，或者说在移动设备的小屏幕上显得太过突兀。

CSS3：扩展em的rem

在处理复杂的响应式布局时，Web设计师通常会面临一个难题。等比缩放的文字单位，比如em和百分比，会使文字相对于包含它们的元素缩放。这在前面那个简单的例子中不是问题，因为包含元素是容纳整个页面的<body>，或者另一个继承了<body>元素字体设置的元素。可是，在对多级嵌套的布局应用等比缩放时，麻烦就来了。

比如，你把第一层<div>的文字大小设定为1.1 em，把这层<div>中的<h1>设定为2 em。你心里想的是这个标题的文字大小是默认文字大小的两倍，而实际上呢，这个标题文字的大小却是容器文字大小1.1 em的两倍。简单一乘，可知这个标题文字大小最终是默认文字大小的2.2倍。

要避免这种复合的效果，就得始终记着自己在什么地方设定过文字大小。理想情况下，当然是只在一层布局元素上设定。不过，CSS3引入了一个新的单位，非常巧妙地解决了这个问题，这个单位就是rem（root em）。本质上，em和rem都是相对度量单位，但区别在于用rem设定的文字大小始终相对于<html>元素（而不是包含元素）的文字大小计算。因此，2 rem永远等于布局默认文字大小的两倍，在哪一级布局上都是。

目前浏览器对rem的支持非常好，当然，IE7和IE8又不在此列。虽然技术上可以使用JavaScript腻子脚本填补这个空白（参见http://tinyurl.com/rem-polyfill），但大多数Web开发者都很明智，不愿意仅仅为了一个度量单位而让页面又多加载一个脚本，因此至今仍然坚持只使用em。

当然，排版工作不仅仅是设定文字大小。要保证在各种屏幕上的可读性，还要考虑行长、行高、外边距，甚至要使用多栏分隔文本（像6.4.6节介绍的那样）。普通的流式布局和比例缩放很难解决上述这些问题。因此，我们得换一种思路：创建更灵活的样式表，使用媒体查询有针对性地微调这些细节，稍后我们会讲到。但首先，你得理解另外一个影响因素，那就是手机的自动缩放行为。

7.1.4 理解视口

理论上，我们前面介绍的两栏布局可以适应任意窗口大小。但实际上，对于小屏幕移动设备而言，情况还要复杂那么一点点，那就是还要考虑**视口**（viewport）大小。

苹果公司在推出iPhone的时候，为了让iPhone的小屏幕尽可能完整显示当时的网站，发明了“视口”这个概念。当时的网站设计师还不知道什么是响应式设计呢。有了视口的概念之后，iPhone上的Safari浏览器就可以显示更多的网页内容，而不仅仅是显示网页的一个角落。这个能缩小显示网页的区域，就叫视口。

视口这种技术是权衡折衷的产物。它可以保证网页在手机上看起来更像在桌面浏览器中的样子。可这样一来，网页中的文字就小得难以辨认了。用户滚动页面的操作减少了，但放大缩小的操作增加了。用户能更容易找到网页中自己关注的部分，但要阅读其中的内容就会比较麻烦。

注意 苹果引入视口的概念后，所有移动开发者也都认同了这个做法。唯一的区别就是设定多大的视口，以及显示页面的多大部分。

如果你开发的网站只针对桌面浏览器，那不用管设备的视口。视口的默认设置会确保网站在超小的屏幕上也能显示得有模有样（尽管用户可能并不会觉得缩小后的网站非常方便）。可是，如果你想做的就是响应式设计，就是要设计一个真正的、移动设备友好的网站，那就得修改视口的设置。你得告诉浏览器不要自动执行视口缩放，很简单，只要像下面这样在页面的`<head>`部分加一个`<meta>`元素即可：

```
<meta content="initial-scale=1.0" name="viewport">
```

这行代码告诉移动浏览器使用真实的页面比例，不要缩放。比如，对iPhone来说，这就意味着网页要显示在一个320像素宽的屏幕上，而且不缩放。如果不修改视口设置，iPhone会按照980像素的桌面级宽度来显示你的页面，然后再缩小显示在视口中。图7-4展示了上述区别。



图7-4：左：iPhone默认会自动缩小页面，让这个流式布局像是显示在桌面浏览器中一样。结果，此时的文字非放大无法看清。右：禁用缩放之后，可以看到页面在小屏幕中的真实效果。下一步就是使用媒体查询简化布局在小屏幕中显示时的效果

注意 读者可能已经知道了，有不少在线模拟器可以让你看到自己的网站在不同移动设备中的样子。比如，http://mobiletest.me可以让你在最新的iPhone、iPad和Android设备上比较网站的外观。不过，多数模拟器都不支持自动缩放。换句话说，在这些模拟器中预览网站时，网站看起来会像你在<meta>标签里跟上面似的把初始比例设置为1的效果一样。如果你并没有像上面那样设置，那模拟器中的结果就不可信了，你得多加注意。

7.2 使用媒体查询适配布局

前面介绍了怎么创建流式的、可以缩放适应任何浏览器窗口的布局。这种布局能保证页面在任何窗口中正常显示，但却不能保证它们总是赏心悦目。

简单的流式布局在某些极端情况下可能很难看。比如，在非常小的窗口中，多栏布局中每一栏的宽度将极其有限，此时的文字和图片会挤作一团，完全没有可读性。而在非常大的窗口中，每一栏又会变得极宽，由于每一行文字都很长很长，读者看完一行，再找下一行就会变得很困难。

对于上述问题，可以限定布局的最大宽度和最窄宽度。一般是使用max-width和min-width这两个属性。如果窗口宽度超出了最大宽度，则超出部分会以外边距填充。反之，如果窗口宽度缩小到小于最小宽度，则相应的栏会保持最小宽度不再缩小。这样，从某种程度上可以实现对布局视觉效果的保护。但从另一方面看，限定宽度无疑会削弱响应式设计的价值。如果把最小宽度设

定得比iPhone窗口宽，那么iPhone用户看起来就会不爽。

显然，更好的办法是随着窗口大小变化渐进地调整布局结构。比如，在小窗口里就使用不带侧栏或广告栏的单一流线布局，而在大窗口中则可以增大字号并使用多栏布局（参见6.4.6节）。

媒体查询（media query）正是为此而生的。CSS3的这个功能可以让我们根据不同的浏览需求，分别设计不同的样式。使用适当的情况下，通过它可以实现不改动一行HTML就让布局既能适应超宽的屏幕，又可以满足iPhone等手持设备的需要。

7.2.1 媒体查询

媒体查询必须知道查看网页的设备的某些重要信息（比如屏幕大小、分辨率、颜色位深，等等）。根据这些信息，可以分别应用不同的样式甚至替换整个样式表。

最简单的媒体查询在样式表里就是一个独立的代码块。这个代码块以`@media`开头，随后是一对圆括号，然后是一对花括号，其中包含符合条件的情况下要应用的样式。媒体查询的基本结构如下：

```
@media (media-feature-name: value) {
  /*符合条件时应用的样式*/
}
```

显然，媒体查询很像JavaScript中的条件块。如果浏览器当前的条件与圆括号中的条件匹配，它就会采用花括号中的那些样式。如果不匹配，浏览器会忽略这些样式。

注意 浏览器始终会采用位于`@media`代码块之外的样式。满足条件时的媒体查询样式是在其他样式基础之上应用的。为此，条件式媒体查询样式经常要覆盖其他样式，比如隐藏之前可见的元素，把某个区块移动到另一个位置，调整字号大小，等等。

7

使用媒体查询之前，必须知道可以“查询”媒体（设备硬件）的哪些条件。媒体查询标准规定了可以查询的各种信息，这些信息被称为媒体特性（media feature）。比如，可以在显示区域的宽度缩小到一定值的时候改变样式。表7-1列出了最常用的媒体特性。（这个表里没有包含那些尚未得到广泛支持的特定于浏览器的试验性媒体特性。）

表7-1 媒体查询中最常用的媒体特性

特性名	值	应用场景

`width`
`min-width`
`max-width`
	显示区域(对打印机而言是打印表面）的宽度	改变布局以适应非常窄（如手机）或非常宽的显示器

`height`
`min-height`
`max-height`
	显示区域的高度	改变布局以适应非常长或非常短的显示器

`device-width`
`min-device-width`
`max-device-width`
	当前计算机或设备屏幕的宽度（或打印输出时纸面的宽度）	根据不同设备（如手机）调整布局

（续）

特性名	值	应用场景
device-height min-device-height max-device-height	屏幕或纸面的高度	根据不同设备（如手机）调整布局
orientation	landscape（横向）或portrait（纵向）	根据设备的朝向调整布局
device-aspect-ratio min-device-aspect-ratio max-device-aspect-ratio	显示区域的宽高比（1/1是正方形）	根据窗口形状调整样式（问题可能比较复杂）
color min-color max-color	屏幕颜色的位深（1位表示黑白，目前主流显示器都是24位，每个像素可以显示数百万种颜色）	检查是否支持彩色输出（比如，是不是要黑白打印），或者支持的颜色数量

CSS媒体类型

CSS的制定者们在CSS2.1中尝试解决过多设备的问题。当时引入了所谓的“媒体类型”，估计读者中有人用过它单独为打印提供样式表：

```
<head>
  ...
  <!-- Use this stylesheet to display the
       page onscreen. -->
  <link  rel="stylesheet" media="screen"
    href="styles.css">

  <!-- Use this stylesheet to print the
       page. -->
  <link rel="stylesheet" media="print"
    href="print_styles.css">
</head>
```

这里的media属性的值还可以是handheld（手持），即低速、小屏的移动设备。因此，现代的移动浏览器大都会忽略这种针对手持设备的样式，导致media属性难以胜任当今各种移动上网设备的需要。但不管怎么说，通过它提供打印样式表还是个不错的方式。

7.2.2 创建简单的媒体查询

你可能注意到了，多数媒体查询特性都允许指定最大或最小限制。这些限制很重要，因为多数媒体查询适用的值很宽泛。

使用媒体查询之前，首先要选择你想检测的属性。比如，要针对窄屏窗口设置一组样式，就要选择max-width。之后，你可以随意设置限制。看一个例子，下面这个媒体查询块中的样式会在浏览器窗口小于等于480像素时应用：

```
@media (max-width: 480px) {
  ...
}
```

提示 目前最流行的媒体特性是max-device-width（用于创建手机版网站）、max-width（用于针对窗口宽度设定不同的样式）和orientation（用于根据平板电脑或iPad的横向或竖向来改变布局）。

要测试某个媒体查询，可以通过它应用一种明显的变化。比如，可以通过媒体查询修改一栏的背景颜色：

```
@media (max-width: 480px) {
  .leftColumn {
    background: lime;
  }
}
```

接下来检查这个媒体查询是否有效。缓慢缩小浏览器窗口，等缩小到480像素时，新的样式就会起作用，左栏的背景变成黄绿色。与此同时，给左栏设定的其他样式仍然有效（比如它的位置和大小），媒体查询样式不会覆盖它们。

注意 不理解媒体查询的浏览器（如IE8）会忽略这些新样式，窗口变大或缩小不会影响原有样式。

如有必要，可以再加一条媒体查询，在屏幕更窄时应用新样式。比如，下面这条媒体查询中的样式会在窗口缩小到250像素时起作用：

```
@media (max-width: 250px) {
  ...
}
```

只要知道一点：新规则会叠加到原有规则之上。换句话说，这里的样式会叠加到常规样式和450像素媒体查询样式之上，最终得到的是这些样式混合后的效果。有点糊涂？没关系，稍后我们还会介绍怎么更细化地定义媒体查询。接下来我们还是先看一个实用的例子吧。

7.2.3 构建移动设备友好的布局

有了媒体查询，就可以创建出能够适应手机浏览器和桌面浏览器的网站。而我们要做的，就是定义不同的样式。

图7-5展示了第2章的ApocalypseSite.html页面的一个修改版。原来的页面是固定布局，栏宽是写死的。这个新版本则使用了本章介绍过的所有新技术，包括适应任意窗口宽度的流式布局。而且，还使用了em单位设定了外边距、内边距、边框宽度、字体大小，以保证这些细节能随设备屏幕变化相应调整。页面顶部的图片也可以放大或缩小，以填充可用空间。左栏中的广告图片则应用了流式图片技术，从而不会超出栏边界。当然，这个页面还通过<meta>元素设定了页面不能在移动浏览器中放大。



图7-5：这个页面采用了响应式Web设计的最佳实践。访问示例网站可以查询完整的CSS：http://prosetech.com/html5

简言之，这个新的ApocalypseSite.html页面考虑了移动设备，但还称不上移动设备友好。为什么这么说呢？因为不管并排的两栏怎么变窄，它们在小屏幕中看起来都不好看。因此，就要用到媒体查询了。

动手写样式之前，应该先确定移动版网站的外观。一般来说，移动版网站都只有一栏，原来的侧栏要么隐藏，要么转移到主内容之上或之下。图7-6展示了ApocalypseSite.html页面在iPhone中的一种外观：

创建移动版ApocalypseSite.html页面非常简单。页眉和文字会自动缩小，因为我们使用的是流式布局和em单位。剩下的就是使用媒体查询重新排列分栏了。

开始的时候，页面中的两栏由以下样式规则控制：

```
.NavSidebar {
  float: left;
  width: 22%;
  font-size: small;
}

.Content {
  float: left;
  width: 78%;
}
```



图7-6：这里是ApocalypseSite.html页面移动版的两个截图。页面顶部是一个小页眉，紧接着是文章内容（左图）。原来左侧中的文章链接和广告转移到了文章内容的下方（右图）

左栏被设定为22%宽，向左浮动。内容栏被设定为78%宽，浮动到左栏右侧。

因为这个布局在浏览器窗口变窄后就不好看了，所以应该使用流行的max-width媒体特性。如前所述，max-width会取得当前浏览器窗口的宽度。如果这个值比较小，那么并排的两栏就很难看。

以下就是去掉浮动、重设栏宽（每栏占满可用宽度）的媒体查询：

```
@media (max-width: 568px) {
  .NavSidebar {
    float: none;
    width: auto;
  }

  .Content {
    float: none;
    width: auto;
  }
}
```

这些样式会叠加在已有样式之上。因此，有必要的话就得把修改过的样式重置回其默认值。具体来说，这里的媒体查询把float属性重置为none，把width属性重置为auto（重置为100%也可以）。这些都是默认值，最初的分栏样式修改了这些值。当然，最初的NavSidebar样式也设定了文字大小，但这里的媒体查询并没有重置该样式，因此该样式依旧起作用。

从技术角度讲，这里的媒体查询定义的是窄窗口中页面的样式，无论用户使用移动浏览器还是桌面浏览器的窄窗口，看到的效果是一样的。这样很好，但在此基础上，我们还可以为桌面浏览器的窄窗口和移动浏览器分别设定不同的样式。要针对桌面浏览器的窄窗口，应该使用max-width，而要针对移动浏览器，就要使用max-device-width，如表7-1所示。

提示　窗口多大时切换到简化布局并没有一定之规，但568像素是个不错的选择，因为568像素是iPhone横向时的页面宽度（对Android设备也一样，稍后我们会解释）。

这个例子还需要再调整一次。在页面初始版的HTML中，NavSidebar位于Content之前，这是为了方便将两者都向左浮动，而且NavSidebar会位于左侧。但在移动版中，去掉浮动之后，由于标记会按先后次序渲染，最终NavSidebar会出现在页面顶部，把Content压在下面。对移动用户而言，这种布局有点本末倒置，因为他们必须先向下滚动页面，在一堆链接和广告之后才能看到正文内容。

针对类似的问题，解决方案是从一开始就不要像原始版那样排布标记，而是先考虑移动版本需要，然后再用额外的CSS规则创建想要的多栏布局。这种思路就是所谓的“移动优先”设计。

具体到眼下这个例子，就是在HTML中把Content放在NavSidebar前面。这样就可以解决移动版本中的页面布局问题，但同时也会导致边栏在完整版中居右。怎么办？很简单，只要向右浮动Content即可：

```
.Content {
  float: right;
  width: 78%;
}
```

这样，Content依然位居右侧，NavSidebar还是左栏，与图7-5所示的完整版布局毫无二致。

此时，我们还得考虑再添加一个媒体查询块，针对非常宽的浏览器窗口写一些样式。比如为保证文本的可读性，将正文文字分成多栏（使用6.4.6节介绍的技术）。

提示　想看一些有启发性的例子？那好，推荐你关注现成的响应式模板。网上有很多，比如http://html5up.net、www.typeandgrids.com，还有http://responsify.it。

7.2.4　媒体查询的高级条件

有时候，我们可能想通过媒体查询设定更具体的样式。这就需要用到更多条件了，比如：

```
@media (min-width: 400px) and (max-width: 700px) {
  /* These styles apply to windows from 400 to 700 pixels wide. */
}
```

这种媒体查询块非常适合几组互相排斥的样式，又不会带来令人头疼的样式分层问题。下面是几个例子：

```
/* Normal styles here */

@media (min-width: 600px) and (max-width: 700px) {
  /* Override the styles for 600-700 pixel windows. */
}

@media (min-width: 400px) and (max-width: 599.99px) {
```

```
  /* Override the styles for 400-600 pixel windows. */
}

@media (max-width: 399.99px) {
  /* Override the styles for sub-400 pixel windows. */
}
```

如果浏览器窗口是380像素，就会应用两组样式：标准样式和最后一个@media块中的样式。这种方式到底会使开发工作简化还是复杂化，取决于你想实现的结果。如果样式原本就复杂，而修改又很多，那这里展示的非叠加方式通常是最简单的方式。

注意，你需要自己注意样式规则不会意外叠加。如果你在一条规则里设定的最大宽度是400像素，而在另一条规则设定的最小宽度同样也是400像素，那这两条规则中的样式就可能叠加。有一个不那么优雅的方案，就是使用小数，比如在其中一条规则写399.99像素。

还有一个方案，就是使用not关键字。两个方案在功能上没有差别，你喜欢哪个就用哪个吧：

```
/* Normal styles here */

@media (not max-width: 600px) and (max-width: 700px) {
  /* Override the styles for 600-700 pixel windows. */
}

@media (not max-width: 400px) and (max-width: 600px) {
  /* Override the styles for 400-600 pixel windows. */
}

@media (max-width: 400px) {
  /* Override the styles for sub-400 pixel windows. */
}
```

隐藏和替换<section>

如果你有想法，那可以把移动版改得更好，让它与桌面版差别更大。比如，可以使用CSS的display属性隐藏或显示页面的整个<section>。

但在动手之前，我想先说说这么做的坏处。切换页面中较大的<section>可能导致标记混乱，不好维护，因此为保证一致性，必须在不同设备中测试。同样，如果隐藏的分区里包含图片，浏览器仍然会下载它们，只是不显示而已。对移动设备来说，这无疑会导致性能问题，同时还浪费流量。

不过，还是有很多情况适合使用分区切换技术的。比如，把复杂的导航或菜单，替换成轻量简单的移动版。最常见的，就是在移动版里用下拉菜单代替展开的树状导航。（有人甚至还为此写了一个样式，可以把一行链接转换成下拉列表，详见：http://css-tricks.com/convert-menu-to-dropdown。）

有时候，简单的调整和替换还不够，你可能想让移动版的变化更大一些。极端情况下，你可以创建一个单独的移动网站，托管在一个不同的域名下（比如《纽约时报》的移动版：http://mobile.nytimes.com）。这样做的工作量最大，如果不使用内容管理系统（CMS），很难保

证移动网站与标准网站同步。还有一种策略，就是通过服务器端代码检测每一个请求，确认浏览器来自移动设备还是桌面设备，然后分别发送不同的内容。这种方案更好，当然对你的时间和技能也是个挑战。

比较好接受的方式，则是使用JavaScript根据浏览器不同动态修改页面。比如，可以使用Modernizr提供的Modernizr.mq()方法，测试代码中的媒体查询特性（参见http://modernizr.com/docs）。这种方法比媒体查询更好，可是也增加了页面设计的复杂度。

在上面的例子中，还是存在样式叠加的情况。这是因为@media块要建立在标准的非媒体查询的样式规则之上。建议大家把这两部分样式分开保存（比如专门将适用于移动设备的样式保存为一个文件）。而分开保存之后，就需要针对外部样式表的媒体查询了。

7.2.5 替换整个样式表

如果样式修改不多，使用@media块就行，所有样式都可以放在一个文件里。但如果要修改的地方很多，那可能创建一个新样式表会更便于管理。创建新样式表之后，可以使用媒体查询创建一个样式表链接：

```
<head>
  <link rel="stylesheet" href="standard.css">
  <link rel="stylesheet" media="(max-width: 568px)" href="small_styles.css">
  ...
</head>
```

这样，浏览器会下载第二个样式表（small_styles.css），但只有在页面宽度小于568像素时才会应用这个样式表。

跟前面的例子一样，此时的新样式会与已经存在的样式叠加。某些情况下，可能需要让几个样式表完全独立，不叠加。此时，就要为标准样式表添加一个媒体查询，保证它在大窗口下起作用：

```
<link rel="stylesheet" media="(min-width: 568.01px)" href="standard.css">
<link rel="stylesheet" media="(max-width: 568px)" href="small_styles.css">
```

这种方式的问题在于，不理解媒体查询的浏览器会忽略这两个样式表。为了兼容旧版本IE，可以使用条件注释引入标准样式表：

```
<link rel="stylesheet" media="(min-width: 568.01px)" href="standard.css">
<link rel="stylesheet" media="(max-width: 568px)" href="small_styles.css">
<!--[if lt IE 9]>
  <link rel="stylesheet" href="standard.css">
<![endif]-->
```

这个例子还存在一个盲区。旧版本Firefox（3.5之前）既不理解媒体查询，也不理解IE的条件注释。可以通过代码检测浏览器，然后再用JavaScript切换页面，但这样有点乱。所幸的是，旧版本的Firefox越来越少见了。

偶尔，也可以试着组合使用媒体查询和媒体类型。但要记住，一定要把媒体类型放在前头，

而且别把它放在括号里。下面是两个打印样式表的例子：

```
<link rel="stylesheet" media="print and (min-width: 25cm)"
href="NormalPrintStyles.css" >
<link rel="stylesheet" media="print and (not min-width: 25cm)"
href="NarrowPrintStyles.css" >
```

7.2.6 识别特定的移动设备

如前所述，使用max-device-width可以区别普通计算机和移动设备。但这个宽度到底设成多大合适呢？

如果想检测手机，那么将max-device-width设定为568像素。这是个经验值，涵盖目前的iPhone和Android手机，而且不管是竖向还是横向：

```
<link rel="stylesheet" media="(max-device-width: 568px)"
  href="mobile_styles.css">
```

如果你对硬件很了解，可能会觉得这条规则太简单粗暴了。毕竟，当今的移动设备都使用超高分辨率的屏幕。比如，iPhone 5的分辨率是640像素 × 1136像素。因此，你可能觉得应该设定更大的宽度来识别这些设备。但实际上却不是那么回事。

还是以iPhone 5为例，尽管它的物理像素是640像素，但它会声明自己的CSS像素为320像素（竖向）。这样是为了避免让网站误以为自己是大屏幕，而显示桌面版的网站。iPhone当然可以显示桌面版网站，但像素那么小，显示出来也没法看清。

最新的高分辨率设备都是这样处理了。它们为此引入了一个比例，叫像素比（pixel ratio）。比如，iPhone（4及之后的型号）用两个物理像素对应一个CSS像素，因此像素比是2。因此，可以通过下列媒体查询来识别iPhone 4：

```
<link rel="stylesheet"
media="(max-device-width: 480px) and (-webkit-min-device-pixel-ratio: 2)"
href="iphone4.css">
```

表7-2列出了几种流行设备的宽度，这里面包含了像素比的概念。比如，所有iPad都会报告自己的宽度是768，但iPad 3的物理像素宽度其实两倍于此。

表7-2　常见设备宽度

设　　备	竖向宽度	横向宽度
iPhone 4	320	480
iPhone 5	320	568
iPad	768	1024
三星Galaxy S4	360	640
谷歌Nexus 4	384	640
Kindle Fire	600	1024

提示　市面上的新设备层出不穷，要了解最新信息，请参考相关网站。

iPad等平板电脑的问题更多，因为用户经常会改变方向。改变方向虽然会改变max-width，但不会改变max-device-width。无论竖向还是横向，iPad始终报告自己的设备宽度为768像素。好在，我们可以组合使用max-device-width和orientation属性，以便区别iPad的方向应用不同的样式：

```
<link rel="stylesheet"
  media="(max-device-width: 768px) and (orientation: portrait)"
  href="iPad_portrait.css">

<link rel="stylesheet"
  media="(max-device-width: 768px) and (orientation: landscape)"
  href="iPad_landscape.css">
```

当然，这条规则不仅限于iPad，其他屏幕大小类似（768像素或更小）的设备也适用。

注意 只有媒体查询不足以将常规网站转换成移动友好的网站。考虑到用户体验，你可能还得把内容切分成小块（减少滚动），去掉那些在触摸屏上难以控制的效果和交互手段（如弹出菜单）。

媒体查询与视频

显示视频是桌面网站与移动网站的一个主要区别。移动网站当然可以包含视频，但窗口会更小，而且文件也会更小。一方面移动浏览器速度要慢一些，流量也不便宜，另一方面回放的硬件也没有桌面电脑那么强。

使用我们学到的媒体查询技术，可以方便地修改<video>元素的大小，适应移动用户的需求。但接下来才是关键：怎么链接到切分后的小视频文件？

HTML5提供了一个方案，即使用<source>元素的media属性。第5章讲过，<source>元素用于指定<video>元素播放的媒体文件。通过添加media属性，可以限制某些设备只能下载对应的文件。

下面的例子展示了怎么让小屏幕设备下载butterfly_mobile.mp4，让其他设备下载butterfly.mp4或butterfly.ogv（看支持哪种视频格式）：

```
<video controls width="400" height="300">
  <source src="butterfly_mobile.mp4"
   type="video/mp4"
   media="(max-device-width: 480px)">
  <source src="butterfly.mp4"
  type="video/mp4">
  <source src="butterfly.ogv"
  type="video/ogg">
</video>
```

你可以针对移动用户将视频重新编码。编码工具一般都有针对设备的配置文件。比如，可能会有一个选项叫“iPad视频”。另外，视频的格式也取决于你自己（H.264很常用）。

第 8 章

基本Canvas绘图

正如我们在第1章介绍过的，HTML5的目标之一就是让网页中的富应用实现起来更简单。当然，这里所谓的“富”指的可不是你银行账户中的钱。富应用的含义包括漂亮的图片、人机互动功能，以及眩目的动画效果。

Canvas是实现富应用最重要的HTML5工具之一，这块“画布”能够把你内心深处的“毕加索”释放出来。与其他HTML元素相比，<canvas>独特的地方是需要JavaScript来操作。不使用JavaScript，就无法绘制图形，也不能画出图画。这也就意味着<canvas>是一个编程工具，而这已然超出了Web基于文档的设计初衷。

表面上看，使用<canvas>似乎就是把简化版的Windows“画图”程序硬塞到了网页里。但深入之后，你会发现这个元素是一切高级图形应用的核心所在。利用它，可以开发出很多你梦寐以求的东西（比如游戏、地图和动态图表），也可以开发出你从未想过的东西（比如音乐灯光秀、物理模拟器）。在不远的过去，要开发出这些东西，如果没有Flash等插件是极其困难的。而今天，有了<canvas>，这一扇门终于敞开了。只要你愿意，这些作品就可以从你的手中创造出来。

本章，我们学习如何在页面中添加<canvas>，并在其中绘制线条、曲线和简单图形。然后，学以致用，开发一个简单的绘图程序。另外，大概也是最重要的，我们会讨论怎么让包含<canvas>的页面在不支持HTML5的旧浏览器中正常运作。

8

注意 <canvas>对某些开发人员来说是不可或缺的，而对另一些人来说可能只是一种消遣。（还有一些人对<canvas>感兴趣，但他们会觉得与使用Flash等成熟的编程平台相比，学习这门新技术有点麻烦。）不过有一件事是肯定的：这个直观的绘图界面对百无聊赖的程序员而言，绝非一个玩具那么简单。

8.1 Canvas 起步

<canvas>元素就是一块画布，就是你提笔挥洒写意的地方。从标记的角度看，它简单明了，只要给它指定三个属性即可：id、width和height。

```
<canvas id="drawingCanvas" width="500" height="300"></canvas>
```

其中，id属性是一个唯一的名字，JavaScript脚本可以利用它找到这块“画布”。相应地，width和height属性指定的就是这块“画布”的宽度和高度，单位是像素。

注意 一定要通过width和height属性设置<canvas>的宽和高，而不要在样式表中设置其宽度和高度。9.1.1节的附注栏解释了如果通过样式表设置<canvas>的宽和高会导致什么问题。

开始的时候，<canvas>在页面上会显示一块空白、无边框的矩形（意思就是你看不到它）。为了让它在页面显现出轮廓，可以通过一条样式规则为它应用不同的背景颜色或者边框：

```
canvas {
  border: 1px dashed black;
}
```

图8-1展示了这块空白的画布。

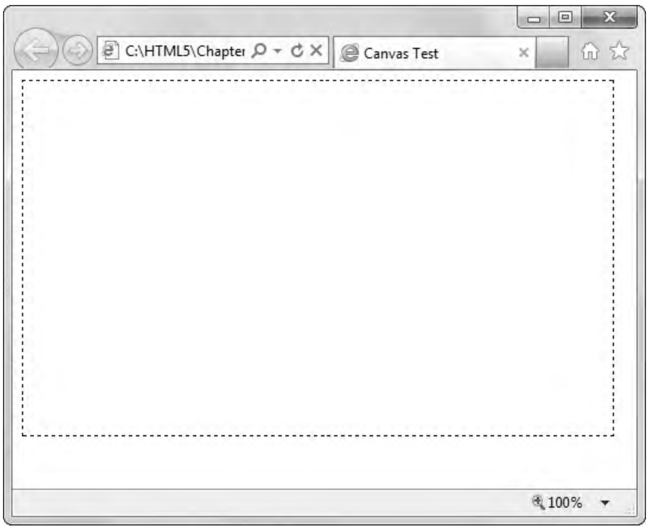


图8-1：每个<canvas>一开始就是一个空白的矩形。哪怕要在上面画一条直线，你都得编写JavaScript代码

开始绘图之前，需要JavaScript执行两步操作。首先，利用document.getElementById()方法取得<canvas>对象。这个例子中的<canvas>对象ID是drawingCanvas，代码如下：

```
var canvas = document.getElementById("drawingCanvas");
```

这没有什么需要解释的，当你需要从页面中取得某个HTML元素时，就要使用document.getElementById()方法。

注意 如果你不熟悉JavaScript，也不要着急。学会基本的JavaScript并不难，看看附录B 。

拿到了<canvas>对象，就可以开始第二个必需的步骤，调用<canvas>对象的getContext()方法，取得二维绘图上下文：

```
var context = canvas.getContext("2d");
```

什么是绘图上下文？你可以把它想象成一个超级强大的绘图工具，它可以帮你完成所有绘图任务，比如绘制矩形、输出文本、嵌入图像……总之，所有绘图操作都是通过它来完成的。

注意 这里的上下文明确地称为“二维上下文”（在代码中用2d表示），可能就会有读者想问：那有没有三维绘图上下文呢？目前还没有，但HTML5的制定者正在考虑，将来就会有的。

取得了上下文对象之后，任何时候都可以进行绘图了。比如，可以在页面加载完毕后、用户单击了按钮时，或者其他时机。刚开始接触<canvas>的读者，心里可能会想：要是能有一个直观的练习页面就好了。好吧，下面就是那么一个模板页面：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Canvas Test</title>
  <style>
canvas {
  border: 1px dashed black;
}
  </style>

  <script>
window.onload = function() {
  var canvas = document.getElementById("drawingCanvas");
  var context = canvas.getContext("2d");

  //（把你自己的绘图代码写在这里）
};
  </script>
</head>

<body>
  <canvas id="drawingCanvas" width="500" height="300"></canvas>
</body>
</html>
```

页面的<body>标签内只有<canvas>元素，没有别的。代码中的<style>部分为<canvas>加了边框，以便它在页面中显示出轮廓来。而<script>部分则主要处理window.onload事件，这个事件是在浏览器加载完页面时触发的。然后，代码取得了<canvas>对象，并创建了绘图上下文，为下一步绘图作好准备。就这样了，接下来你就可以用这个页面作为起点开始试验了。

注意 当然，如果是在真实的网站中使用<canvas>，应该把JavaScript代码挪到一个外部文件中，这样才能保持页面的清晰（相关内容请参见附录B）。不过就目前来讲，把所有东西都放在一个页面里可以让试验更方便。建议读者从本书站点（http://prosetech.com/html5）下载CanvasTemplate.html文件，然后自己动手输入下面的示例代码。

8.1.1 画直线

现在一切准备就绪，可以绘图了。等一等，在我们涂鸦之前，还得先了解一个基本知识点：画布的坐标系。图8-2展示了<canvas>中坐标系的概念。

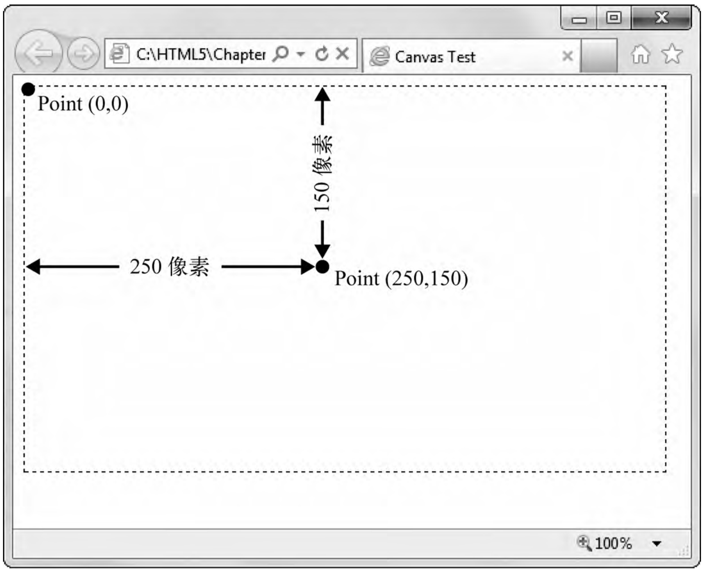


图8-2：与其他HTML元素一样，<canvas>坐标的左上角是坐标原点(0,0)。向右移动，*x*值增大，向下移动，*y*值增大。对于500像素×300像素的<canvas>元素来说，其右下角坐标就是（500,300）

最简单的绘图操作就是画一条实心直线。为此，需要通过绘图上下文执行三个操作。首先，使用moveTo()方法找到直线的起点。其次，使用lineTo()方法在起点和终点之间建立联系。最后，调用stroke()方法，把直线实际地绘制出来：

```
context.moveTo(10,10);
context.lineTo(400,40);
context.stroke();
```

如果你觉得不好理解，也可以这么想：首先，拿起画笔把笔头放在画布上的某一点（使用moveTo方法），然后在画布上画直线（使用lineTo方法），最后让直线显现出来（使用stroke方法）。结果就是一条起点为(10,10)，终点为(400,40)的1像素宽的黑色直线。

不止如此，要是你还有什么创意，也是可以美化直线的。在调用stroke()方法把直线实际地

绘制出来之前，你可以在任何时候设置绘图上下文的3个属性：lineWidth、strokeStyle和lineCap。这几个属性会一直影响后面的绘图操作，除非再修改它们的值。

顾名思义，使用lineWidth可以设置线条宽度，单位是像素。比如，要绘制10像素粗的线条，就要这样设置：

```
context.lineWidth = 10;
```

而strokeStyle用于设置线条的颜色。设置颜色可以使用HTML颜色名、HTML颜色编码或CSS中的rgb()函数。其中，使用rgb()函数可以直接指定红、绿、蓝三个分量的比例。（这种方式很有用，因为很多绘图和平面处理软件都使用RGB颜色表示法。）无论你使用哪种方式，都需要把颜色值放在一对引号内，比如：

```
//使用HTML颜色编码设置颜色（砖红色）
context.strokeStyle = "#cd2828";

//使用rgb()函数设置颜色（砖红色）
context.strokeStyle = "rgb(205,40,40)";
```

注意 之所以把这个属性命名为strokeStyle而不是strokeColor，是因为通过它不仅仅可以设置颜色。下一章我们就会介绍到，通过它还可以设置叫做渐变的混合颜色（参见9.2.3节）和基于图像的图案（参见9.2.2节）。

最后，使用lineCap可以设置线条两端的形状，即线头类型。默认值是butt，即方头。另外，还可以使用round（圆头）或square（效果与butt类似，也是方头，但会在线条的两头各增加一半线宽的长度，因此可以叫“加长方头”。）

以下就是我们要绘制的三条不同线头水平线的全部脚本代码（结果如图8-3所示）。要试验这些代码，可以把它们包装到一个函数里。然后，在前面介绍的window.onload事件处理函数中调用它即可：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

//设置线条宽度和颜色（适用于所有线条）
context.lineWidth = 20;
context.strokeStyle = "rgb(205,40,40)";

//绘制第一条直线，使用默认的方头
context.moveTo(10,50);
context.lineTo(400,50);
context.lineCap = "butt";
context.stroke();

//绘制第二条直线，使用圆头
context.beginPath();
context.moveTo(10,120);
```

```
context.lineTo(400,120);
context.lineCap = "round";
context.stroke();

//绘制第三条直线，使用加长方头
context.beginPath();
context.moveTo(10,190);
context.lineTo(400,190);
context.lineCap = "square";
context.stroke();
```

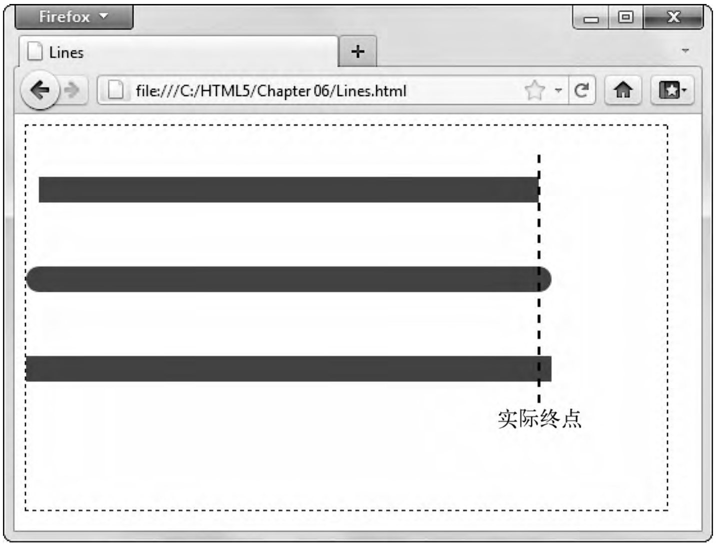


图8-3：上面的线使用标准的方头，而下面的线则使用了加长的线头（一个加长圆头，一个加长方头），即在线条的两头各增加一半线宽的长度

这个例子中又介绍了一个新的特性：绘图上下文的beginPath()方法。每次调用beginPath()方法，都重新开始一个新线段的绘制。如果没有这一步，那么每次调用stroke()，都会把画布上原有的线段再重新绘制一遍。在修改了其他上下文属性的情况下，这个问题会比较明显。就以上面的代码为例，如果不调用beginPath()的话，那么就会发生在原有直线上以新颜色、新宽度或新线头形状重新绘制的问题。

注意 尽管开始绘制新线段时要调用beginPath()，但结束绘制线段则不一定要做什么。每次开始新路径时，原来的路径就会自动“完成”。

8.1.2 路径与形状

为了确保三条直线各自独立，上一个例子将每条直线都按照新路径来绘制。这样可以为不同的直线分别应用不同的颜色（以及不同的线宽和线头）。实际上，路径本身也是很有用的，因为

可以通过路径来填充自定义的形状。例如，以下代码可以绘制出红色的空心三角形：

```
context.moveTo(250,50);
context.lineTo(50,250);
context.lineTo(450,250);
context.lineTo(250,50);

context.lineWidth = 10;
context.strokeStyle = "red";
context.stroke();
```

不过，如果想给这个三角形填上颜色，那么stroke()方法是无能为力的。此时，应该先调用closePath()来明确地关闭路径，然后再把fillStyle属性设置为想要填充的颜色，最后再调用fill()方法完成填充操作：

```
context.closePath();
context.fillStyle = "blue";
context.fill();
```

这个例子还有两个地方有必要调整一下。首先，如果知道最后会关闭路径，那实际上就不必再绘制最后一条线段了，因为closePath()会自动在最后一个绘制点与绘制起点间绘制一条线。其次，最好是先填充形状，然后再绘制其轮廓。否则，形状的轮廓线会有一部分被填充色覆盖掉。

好了，下面就是绘制三角形的完整代码：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

context.moveTo(250,50);
context.lineTo(50,250);
context.lineTo(450,250);
context.closePath();

//填充内部
context.fillStyle = "blue";
context.fill();

//绘制轮廓
context.lineWidth = 10;
context.strokeStyle = "red";
context.stroke();
```

有读者可能已经注意到了，这个例子并没有调用beginPath()方法。这是因为<canvas>在开始的时候，会自动开始一段新路径。如果你想重新开始另一段路径，那就要调用beginPath()。重新开始新路径意味你可能重新设置了线条的样式，或者准备绘制另外一个新的形状。图8-4展示了运行以上代码的结果。

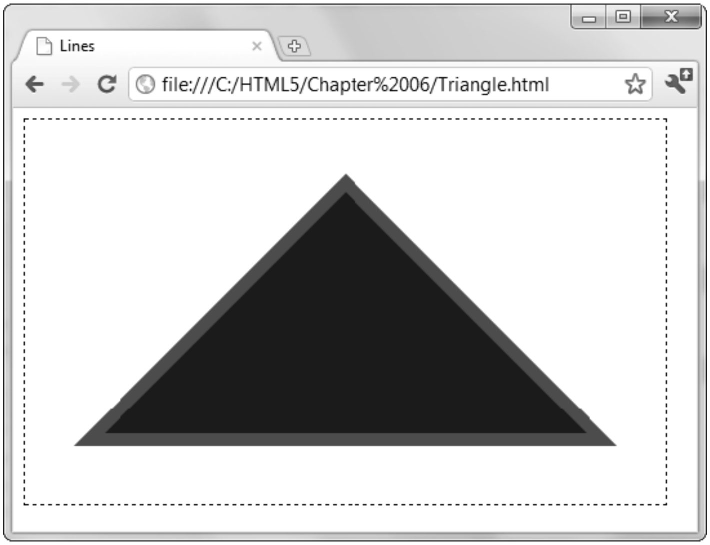


图8-4：要创建一个类似这个三角形的封闭形状，使用moveTo()方法定位起点，使用lineTo()方法绘制每一条线段，然后用closePath()补充完成路径。然后再调用fill()填充，调用stroke()描边

注意 在绘制前后相连的线段时（比如上面例子中的三角形），可以通过设置绘图上下文的lineJoin属性指定线段交点的形状。这个属性的默认值是mitre（锐角斜接），另外两个值是round（圆头）和bevel（平头斜接）。

多数情况下，如果想要绘制复杂的形态，你都需要自己逐个线段地绘制。但有一个例外，那就是绘制矩形。可以使用fillRect()方法直接填充一个矩形区域。只要为它提供矩形区域左上角的坐标、宽度和高度即可。

例如，要在(0,10)点放置一个100像素×200像素的矩形，可以使用以下代码：

```
fillRect(0,10,100,200);
```

与fill()方法一样，fillRect()也是从绘图上下文的fillStyle属性取得颜色。

类似地，还有一个strokeRect()方法，用于直接绘制一个矩形框：

```
strokeRect(0,10,100,200);
```

绘制矩形框时，strokeRect()的宽度取自lineWidth属性，而边框宽度和颜色则取自strokeStyle属性，与stroke()方法一样。

8.1.3 绘制曲线

要是除了矩形和直线之外，你还想弄点别的更有意思的（谁不想呢），那就得理解绘制曲线的四个方法：arc()、artTo()、bezierCurveTo()和quadraticCurveTo()。使用这几个方法分别能够以不同的方式绘制曲线，但共同点是它们都要求你做一点简单的数学计算（有时候计算量还是

蛮大的）。

这四个方法里面，arc()是最简单的，它可以绘制一段圆弧。在画圆弧之前，你可以先闭上眼睛，想象有那么一个圆，而你想绘制的圆弧就是这个圆上的一部分，如图8-5所示。然后，你就可以对要传给arc()方法的参数做到胸有成竹了。

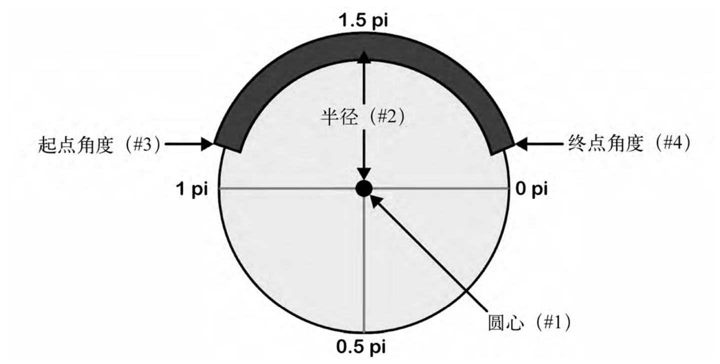


图8-5：圆弧看起来简单，但要描述它得需要多方面的信息。首先，需要确定一个想象中的圆形。而确定圆形就必须有圆心的坐标（#1）和表示大小的半径（#2）。然后，为了描述圆弧的长度，必须知道其起点的角度（#3）和终点的角度（#4）。这两个角度都要用弧度表示，即常量pi的倍数（1 pi是半圆，2 pi是整个圆形）

想通了所有细节之后，接下来就是调用arc()方法：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

//创建变量，保存圆弧的各方面信息
var centerX = 150;
var centerY = 300;
var radius = 100;
var startingAngle = 1.25 * Math.PI;
var endingAngle = 1.75 * Math.PI;

//使用确定的信息绘制圆弧
context.arc(centerX, centerY, radius, startingAngle, endingAngle);
context.stroke();
```

如果在调用stroke()之前调用closePath()，就会在圆弧的起点和终点之间绘制一条直线。于是，就可以得到一个封闭的小半圆。

实际上，圆形也就是这么个圆弧继续向两端伸展构成的。因此如果想画一个整圆，可以这样设置：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

var centerX = 150;
var centerY = 300;
var radius = 100;
var startingAngle = 0;
var endingAngle = 2 * Math.PI;
```

```
context.arc(centerX, centerY, radius, startingAngle, endingAngle);
context.stroke();
```

注意 使用arc()方法画不了椭圆（扁圆）。要画椭圆，要么使用接下来我们会介绍的更复杂的绘制曲线的方法，要么使用变换（参见8.1.4节）把普通的圆拉伸成椭圆。

接下来要介绍的三个方法（artTo()、bezierCurveTo()和quadraticCurveTo()）需要你承受一些几何计算方面的挑战。这三个方法要用到同一个概念：控制点。控制点本身并不包含在最终的曲线里，但能够影响曲线最终的形状。最好的例子就是贝塞尔曲线，几乎任何插图软件中都会用到它。贝塞尔曲线之所以那么流行，就是因为这种曲线能够保证平滑，哪怕再小、再大的弧度都可以。图8-6展示了贝塞尔曲线的控制点。

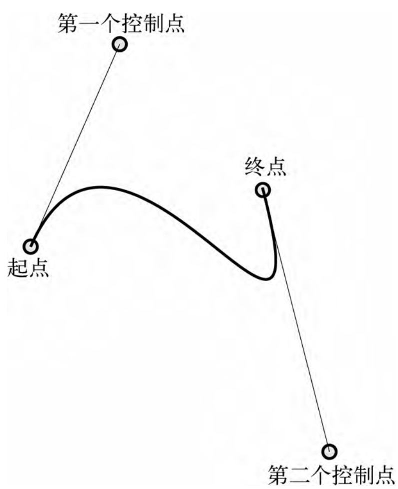


图8-6：一条贝塞尔曲线有两个控制点。曲线的起点切线连接第一个控制点，终点切线连接第二个控制点。两条连接线之间就是曲线。曲线的弯曲程度（曲率）由控制点与起点和终点的距离决定。距离越远，弯曲度越大。这有点像引力，只不过越远力越大

以下就是用于创建图8-6所示曲线的代码：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

//把笔移动到起点位置
context.moveTo(62, 242);

//创建变量，保存两个控制点及曲线终点信息
var control1_x = 187;
var control1_y = 32;
var control2_x = 429;
var control2_y = 480;
var endPointX = 365;
var endPointY = 133;
```

```
//绘制曲线
context.bezierCurveTo(control1_x, control1_y, control2_x, control2_y,
 endPointX, endPointY);
context.stroke();
```

复杂而自然的形状通常需要多个圆弧和曲线拼接而成。完成之后，可以调用closePath()以便填充，或者显示出完成的轮廓。学习绘制曲线的最好方式，就是自己动手编写代码。为此，本书为读者推荐一个不错的测试页面：http://tinyurl.com/html5bezier（如图8-7所示）。

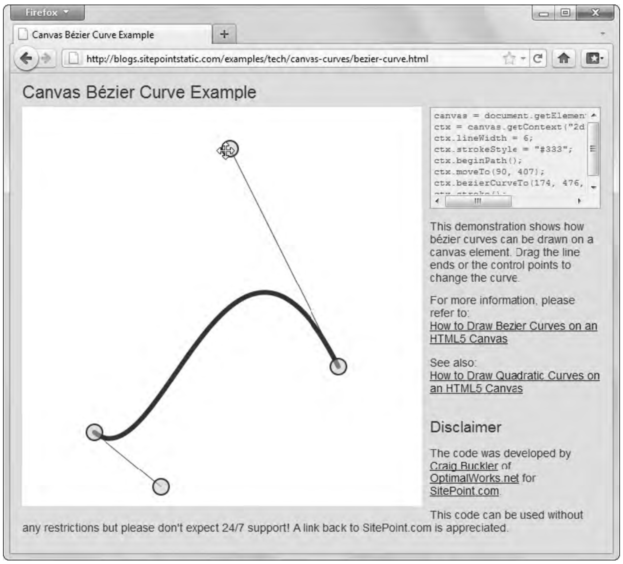


图8-7：在图中所示的这个页面（http://tinyurl.com/html5bezier）里，可以拖动控制点、起点和终点任意改变其中贝塞尔曲线的形状。更重要的是，拖动过程中，页面会为你生成相应的HTML5绘图代码，复制下来就能在自己的页面中创建同样的曲线。这里还有一个所见即所得的二次曲线试验页面tinyurl.com/html5-quadratic

8.1.4　变换

变换，就是一种通过变化<canvas>坐标系达到绘制目的的技术。例如，假设你想在三个地方绘制相同的正方形。为此，可以调用三次rect()，每次都传入不同的起点位置：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

//在三个地方绘制同样大小（30×30）的正方形
context.rect(0, 0, 30, 30);
context.rect(50, 50, 30, 30);
context.rect(100, 100, 30, 30);

context.stroke();
```

或者，也可以在同一个地方调用三次rect()，但每次都移动一下坐标系，最终也能达到在三

个不同位置绘制正方形的目的，比如：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

//在(0,0)点绘制正方形
context.rect(0, 0, 30, 30);

//把坐标系向下、向右各移动50像素
context.translate(50, 50);
context.rect(0, 0, 30, 30);

//把坐标系再向下移一点；变换是可以累积的
//因此现在(0,0)点实际上将被平移到(100,100)
context.translate(50, 50);
context.rect(0, 0, 30, 30);

context.stroke();
```

“数学恐惧症患者”的绘图秘诀

谁能告诉我不用费脑子也能画出这些图形的方法？

如果你希望用<canvas>创作出夺人眼目的图形，但又不愿意复习几何知识，恐怕会有点进退两难。但你是幸运的，下面本书就给你推荐几个方法，让人不用关心数学也能画出自己想要的效果。

- **使用绘图库**。既然有现成的绘图库可以拿来绘制圆形、三角形、椭圆形和多边形，那为啥还自己一笔一笔地画呢？绘图库的设计思路很简单，它们提供很多高级方法（比如，用`fillEllipse()`和坐标值就能绘制椭圆），但底层使用JavaScript帮你完成正确的<canvas>操作。Fabric.js（http://fabricjs.com）和KineticJS（http://kineticjs.com）就是两个绘图库。但这两个（以及更多）库还在快速发展，现在预言哪个库最终能坚持下来还为时尚早，但你可以到目前流行的Stack Overflow网站（http://tinyurl.com/canvas-libraries）读一下关于二者的争论和一些开发者的建议。
- **绘制位图**。与其煞费苦心地在<canvas>上绘图，不如找一张现成的图片嵌入到<canvas>中。比如，要是有一个名为circle.png的圆形图案，可以用9.1.1节介绍的方式把它插入到<canvas>中。不过，插入图片会失去一些灵活性（比如，拉伸、调整或删除其某一部分等）。
- **使用导出工具**。如果图形比较复杂，而你需要在<canvas>上操作它，或者想通过它实现交互，那么绘制位图的方法就不够好了。在这种情况下，如果有一个图形到<canvas>代码的转换工具会比较好。显然，是有这种工具的，比如针对Adobe Illustrator的Ai→Canvas插件（http://visitmix.com/labs/ai2canvas/），能够把Adobe Illustrator的插画转换成HTML网页，并通过JavaScript代码在<canvas>上重新创建相同的图形。

以上两段代码得到的结果都一样：在三个不同位置绘制三个相同的正方形。

表面上看，变换不过就是把一些复杂的绘图任务变得更加复杂了。但在处理一些棘手问题的场合，使用变换却能收到神奇的效果。例如，假设你有一个函数，负责绘制一系列复杂的图形，最终再将它们组合成一幅鸟的图片。现在，你准备让鸟动起来，在<canvas>区域里飞翔。（9.4节将介绍在<canvas>上生成动画。）

如果没有变换，要实现这个目标必须在每次绘制鸟的时候调整一次坐标。而有了变换，绘图代码可以不变，只要反复修改坐标系的位置就好了。

使用变换有几种不同的方式。在前面的例子中，我们使用*平移*（translate）变换移动了坐标系的原点——也就是(0,0)点，默认位置在<canvas>的左上角。除了平移变换之外，还有*缩放*（scale）变换、*旋转*（rotate）变换和*矩阵*（matrix）变换。缩放变换可以把本来要绘制的形状放大或缩小，旋转变换可以旋转坐标系。矩阵变换更复杂一些，但可以在任意方向拉伸和扭曲坐标系，要求你必须理解复杂的矩阵计算，只有这样才能实现自己想要的视觉效果。

变换是累积的。比如，下面这个例子先使用`translate()`方法把坐标系从(0,0)平移到(100,100)，然后又在新位置使用`rotate()`方法把坐标系旋转了几次。每旋转一次，都会绘制一个新的正方形，从而得到如图8-8所示的图形。

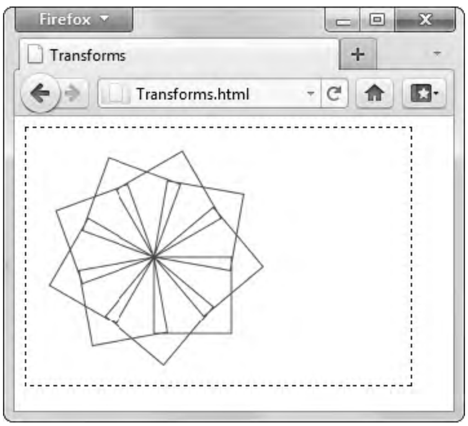


图8-8：通过绘制一系列旋转的正方形，可以生成类似方形螺线的图案

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

//移动(0,0)点。这一步很重要
//因为接下来要围绕新原点旋转
context.translate(100, 100);

//绘制10个正方形
var copies = 10;
for (var i=1; i<copies; i++) {
```

```
    //绘制正方形之前，先旋转坐标系
    //旋转一周是2*Math.PI，因此每个正方形的旋转角度取决于要绘制的总数
    context.rotate(2 * Math.PI * 1/(copies-1));

    //绘制正方形
    context.rect(0, 0, 60, 60);
  }
  context.stroke();
```

提示 调用绘图上下文的save()方法可以保存坐标系当前的状态。然后，再调用restore()方法可以返回保存过的前一个状态。如果要保存坐标系的状态，必须在应用任何变换之前调用save()，这样再调用restore()才能把坐标系恢复到正常状态。而在多步操作绘制复杂图形时，往往都需要多次保存坐标系状态。这些状态就如同浏览器中的历史记录一样。每次调用restore()，坐标系就会恢复到前一个最近的状态。

详细讨论变换超出了本章范围。如果你想全面深入地了解变换，可以参考Mozilla（开发Firefox浏览器的公司）的文档：http://tinyurl.com/canvas-transforms。

8.1.5 透明度

到现在为止，我们一直都在使用实心颜色。实际上，<canvas>支持使用半透明的颜色，从而实现多个形状叠加透视的效果。有两种创建透明图形的方式，第一种就是使用rgba()函数设置透明颜色（即设置fillStyle和strokeStyle属性），而不是使用rgb()函数。注意，rgba()函数接收4个参数：红、绿、蓝颜色分量（0～255）和颜色的不透明度值。最后一个参数（alpha）值为1，表示完全不透明，值为0表示完全不可见。位于0和1之间的值（比如0.5），表示颜色部分透明，即透过它能看到下方的形状。

注意 哪些内容在下，哪些内容在上，完全取决于绘制操作的先后顺序。比如，先画一个圆形，再在相同位置上画一个正方形，则正方形会叠加在圆形上面。

下面这个例子绘制了一个圆形和一个三角形。使用的颜色相同，但三角形的不透明度值被设置为0.5，因此是半透明的：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

//设置填充及描边颜色
context.fillStyle = "rgb(100,150,185)";
context.lineWidth = 10;
context.strokeStyle = "red";
```

```
//绘制圆形
context.arc(110, 120, 100, 0, 2*Math.PI);
context.fill();
context.stroke();

//别忘了调用beginPath()，然后再绘制新形状
//否则，两个形状的路径会意外地连在一起
context.beginPath();

//用半透明的颜色填充三角形
context.fillStyle = "rgba(100,150,185,0.5)";

//好了，绘制三角形
context.moveTo(215,50);
context.lineTo(15,250);
context.lineTo(315,250);
context.closePath();
context.fill();
context.stroke();
```

图8-9是这个例子的结果。

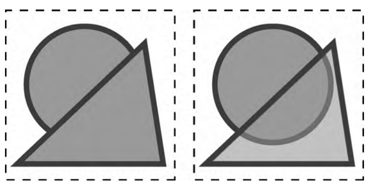


图8-9：左：两个实心、叠加在一起的图形。右：底下的圆形是实心的，上面的三角形是半透明的。半透明的形状看起来较明亮（因为透过它们能看到白色背景），透过它们可以看到下方的内容。注意这个例子中的三角形是半透明的，但三角形的边框则使用了实色

第二种创建透明图形的方式是设置绘图上下文的globalAlpha属性：

```
context.globalAlpha = 0.5;

//此时，再设置的颜色不透明度值都将是0.5
context.fillStyle = "rgb(100,150,185)";
```

这样一来，后续所有绘图操作都会取得相同的不透明度值，也就是会有相同的透明度（直至再次修改globalAlpha属性）。包括描边颜色和填充颜色。

哪种方式更好一些呢？如果你只需要一种透明的颜色，使用rgba()就好了。如果你需要使用不同的颜色绘制很多形状，但每个形状的透明度不一样，可以使用globalAlpha。另外，如果你想在<canvas>上绘制半透明的图像，也要用到globalAlpha属性（参见9.3.2节）。

8.1.6 合成操作

本章到现在一直假设在绘制多个图形时，后绘制的图形会位于先绘制的图形上方，并遮住先绘制的图形。使用<canvas>绘图时，多数情况下都是这样的。然而，<canvas>也支持更复杂的合成操作。

所谓合成操作，就是告诉<canvas>怎么显示两个重叠的图形。默认的合成操作是source-over，即新绘制的图形会位于先绘制的图形之上（即覆盖在第一个图形上）。如果新图形和第一个图形重叠，新的会遮盖第一个 。

注意 在合成操作的术语中，源是指正在绘制的图形，目标是指画布上已经绘制的内容。

还有其他很多种合成方式。比如，可以用xor，它告诉<canvas>不显示两个图形相互重叠的部分。图8-10展示了不同合成操作的结果。

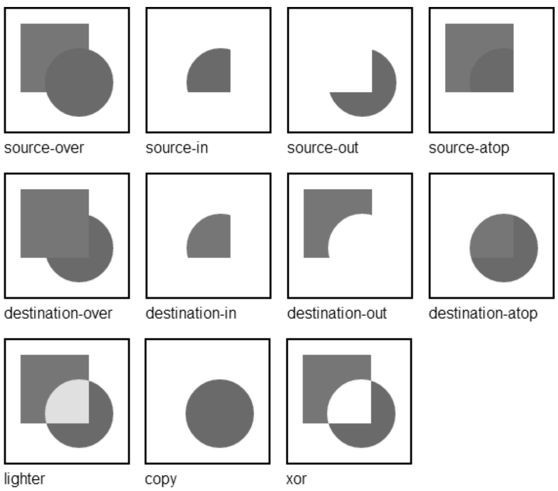


图8-10：这里是11种可能的合成操作及其效果。之前是12种，但不同的浏览器对darker计算的表现不一样，所以这里把它去掉了

要改变<canvas>当前使用的合成操作方式，只要在画后面的图形之前设置绘图上下文的globalCompositeOperation属性即可，代码如下：

```
context.globalCompositeOperation = "xor";
```

比如，要创建图8-10右上角所示的source-atop合成操作，可以用下面这段代码：

```
var canvas = document.getElementById("drawingCanvas");
var context = canvas.getContext("2d");

//绘制矩形
context.fillStyle = "blue";
context.fillRect(15,15,70,70);

//选择globalCompositeOperation
context.globalCompositeOperation = "source-atop";

//在上方绘制圆形
context.fillStyle = "red";
```

```
context.beginPath();
context.arc(75, 75, 35, 0, Math.PI*2, true);
context.fill();
```

只要运用得当，利用合成操作可以迅速实现一些特定的绘图任务，比如绘制复杂的图形。一些厉害的程序员甚至用这些合成操作减少绘图操作次数，以提升绘图性能。

在不久的过去，不同浏览器对某些合成操作的处理方式没有达成一致。幸运的是，现在已不存在这些问题，唯一的问题是要兼容旧版浏览器。现在，唯一支持合成操作的兼容方案是FlashCanvas Pro（参见8.3.2节）。

8.2 构建基本的画图程序

要介绍的<canvas>的功能还有很多。不过，经过本章到现在的学习，我们已经有足够的基础知识构建一个基于<canvas>的画图程序了。图8-11就是本节我们要构建的基本的画图程序。

实现这个程序的JavaScript代码比我们前面看到的所有代码都要长,但实际上却仍然非常好理解。接下来几小节，我们就逐一分析每一段代码。

提示 如果有读者想知道是哪些样式规则创建了图中<canvas>上下方的蓝色工具栏，那么最好是浏览一下完整的代码。如果你只想在自己的浏览器中试验一下这个画图程序，可以打开我们试验站点（http://prosetech.com/html5）上的Paint.html。

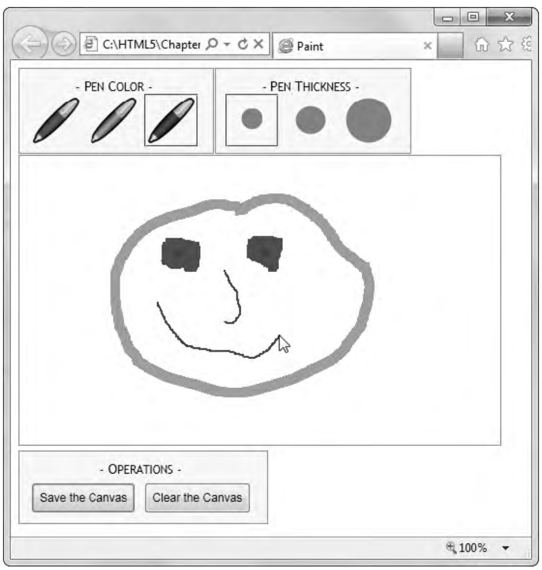


图8-11：要在这个程序中画图，先选择一种颜色，再选择笔画粗细，然后就晃动鼠标吧

8.2.1 准备工作

首先，当页面加载后，脚本代码会取得<canvas>对象，为它添加一些处理函数，以便处理不同鼠标操作导致的JavaScript事件：onMouseDown、onMouseUp、onMouseOut和onMouseMove。（稍后你就会看到，我们正是通过这些事件来控制绘图过程的。）与此同时，代码也把<canvas>保存在了一个全局变量中（变量名为canvas），把绘图上下文保存在了另一个全局变量中（变量名为context）。任何位置的代码都能轻易访问到全局变量：

```
var canvas;
var context;

window.onload = function() {
  //取得<canvas>和绘图上下文
  canvas = document.getElementById("drawingCanvas");
  context = canvas.getContext("2d");

  //添加用于实现绘图操作的事件处理程序
  canvas.onmousedown = startDrawing;
  canvas.onmouseup = stopDrawing;
  canvas.onmouseout = stopDrawing;
  canvas.onmousemove = draw;
};
```

要想开始画图，首先要从窗口顶部的两个工具栏中选择笔画颜色和笔画粗细。这两个工具栏就是两个<div>元素，通过样式给它们添加了熟悉的铁青色背景，还有边框。工具栏中包含一些可以点击的图像（<img>元素）。例如，以下就是供用户选择3种颜色的工具栏的标记：

```
<div class="Toolbar">
  - Pen Color -<br>
  <img id="redPen" src="pen_red.gif" alt="Red Pen"
   onclick="changeColor('rgb(212,21,29)', this)">
  <img id="greenPen" src="pen_green.gif" alt="Green Pen"
   onclick="changeColor('rgb(131,190,61)', this)">
  <img id="bluePen" src="pen_blue.gif" alt="Blue Pen"
   onclick="changeColor('rgb(0,86,166)', this)">
</div>
```

以上标记中的重点在于每个<img>元素的onclick属性。访客单击每幅图像时，都会调用与<img>元素绑定的changeColor()函数。这个函数接收两个参数：与图标颜色匹配的新颜色和对被单击的<img>元素本身的引用。changeColor()函数的代码如下：

```
//记录此前为选择颜色而被单击过的<img>元素
var previousColorElement;

function changeColor(color, imgElement) {
  //重新设置当前绘图要使用的颜色
  context.strokeStyle = color;

  //为刚被单击的<img>元素应用一个新样式
  imgElement.className = "Selected";
```

```
    //恢复上一次被单击的<img>元素的样式
    if (previousColorElement != null) previousColorElement.className = "";
    previousColorElement = imgElement;
  }
```

这个changeColor()函数负责完成两项任务：首先，将绘图上下文的strokeStyle属性设置为新的颜色值。这只需一行代码就够了。其次，改变被单击的<img>元素的样式，即添加实心边框，以便明确显示当前绘图所使用的颜色。这里还需要一点工作，因为要记录上一次选择的颜色，以便将其对应的图像的边框去掉。

接下来的changeThickness()函数几乎与changeColor()完全相同，唯一的区别就是它要修改绘图上下文的lineWidth属性，以保证绘图以适当粗细的笔画进行。

```
  //记录此前为选择粗细而被单击过的<img>元素
  var previousThicknessElement;

  function changeThickness(thickness, imgElement) {
    //重新设置当前绘图要使用的粗细
    context.lineWidth = thickness;

    //为刚被单击的<img>元素应用一个新样式
    imgElement.className = "Selected";

    //恢复上一次被单击的<img>元素的样式
    if (previousThicknessElement != null) {
      previousThicknessElement.className = "";
    }
    previousThicknessElement = imgElement;
  }
```

没错，这些代码没有执行任何实际的绘图操作，我们的例子还没有做完。接下来的（最后）一步，就是添加实际绘图的代码。

8

8.2.2 在画布上绘图

绘图操作从用户在画布上按下鼠标时开始。我们这个画图程序使用了一个名为isDrawing的全局变量，记录绘图什么时候开始，以方便其他代码知悉是否该通过绘图上下文进行绘制。

在前面的代码中，我们看到onMouseDown事件是与startDrawing()函数绑定的。这个函数首先将isDrawing变量设置为true，然后创建新路径，找到起点位置，并作好绘制准备。

```
  var isDrawing = false;

  function startDrawing(e) {
    //开始绘图了
    isDrawing = true;

    //创建新路径（使用当前设置好的描边颜色和线条粗细）
    context.beginPath();

    //把画笔放到鼠标当前所在位置
```

```
    context.moveTo(e.pageX - canvas.offsetLeft, e.pageY - canvas.offsetTop);
  }
```

为了让画图程序正确运行，应该在鼠标当前所在位置开始绘制。鼠标当前所在位置，就是用户在画布上单击鼠标的位置。不过，取得这个位置的坐标还要费点小周折。

onMouseDown事件本身提供了坐标（如代码所示，通过事件对象的pageX和pageY属性），但这两个坐标值是相对于整个页面的。而我们需要的是相对画布左上角的坐标值，所以需要再减去浏览器左上角到画布左上角的距离。

实际的绘图操作要等到用户移动鼠标的时候再开始。用户每次移动鼠标，哪怕只移动了一个像素，都会触发onMouseMove事件并执行draw()函数。此时，如果isDrawing的值为true，draw()函数就会计算当前画布坐标（即鼠标最新位置的坐标），然后调用lineTo()在画布上绘制极小的一段线段，最后再调用stroke()把线条实际地绘制出来：

```
function draw(e) {
  if (isDrawing == true) {
    //找到鼠标的新位置
    var x = e.pageX - canvas.offsetLeft;
    var y = e.pageY - canvas.offsetTop;

    //画一条到新位置的线
    context.lineTo(x, y);
    context.stroke();
  }
}
```

用户继续移动鼠标，draw()函数就会再次被调用，再绘制一小段线段。这条线段非常短，恐怕只有一两个像素，如果用户乱涂的话这看着都不像一条直线。

最后，用户释放鼠标时，或者把光标移动到画布外面时，就会触发onMouseUp或onMouseOut事件。这两个事件都会触发同一个函数：stopDrawing()，这个函数告诉程序停止绘图：

```
function stopDrawing() {
  isDrawing = false;
}
```

关于这个简单的画图程序的代码，到这里已经差不多介绍完了。所剩的就是画布下方的那两个按钮，一个按钮用于保存当前作品，另一个按钮用于清除画布。单击清除按钮，clearCanvas()函数会清空整个画布，使用的是绘图上下文的clearRect()方法：

```
function clearCanvas() {
  context.clearRect(0, 0, canvas.width, canvas.height);
}
```

而保存操作更有意思，因此下一节我们就讨论如何实现保存为图像的操作。

8.2.3 将画布保存为图像

说到把画布保存为图像的方法，那可是太多了。选择什么方法，首先取决于你怎么取得相应的数据。<canvas>元素提供了三个基本的选项。

- **使用数据URL**。把画布转换为一幅图像文件，然后将图像数据转换为字符序列并编码为URL形式。这种方式生成的数据URL非常适合传输图像（例如，可以将数据URL作为`<img>`元素的`src`属性值，或者也可以将其发送到Web服务器）。我们的画图程序就使用这种方法。
- **使用`getImageData()`方法**。取得原始的像素数据，然后可以继续根据需要操作这些数据。关于`getImageData()`方法，我们会在9.5.3节介绍。
- **保存一组"步骤"**。比如，可以把在画布上绘制的每一条线都保存到一个数组中。然后，保存这个数组，以便将来根据该数组重新绘制图像。这个方法不占空间，而且更具灵活性，方便以后编辑图像。但是，前提是你必须记录每一步操作，而相关的技术将在9.3.1节介绍。

如果这些方法已经让你感觉头晕目眩了，请再坚持一小会儿，我们还没结束。确定了要保存什么之后，接下来还要决定保存到哪儿。而这又有三种选择。

- **保存为图像文件**。例如，可以让用户把画布以PNG或JPEG图像格式保存在自己的硬盘上。这也是我们选择的方式。
- **保存在本地存储系统中**。相关内容将在第10章介绍。
- **保存在Web服务器上**。在把数据发送到Web服务器后，服务器端程序可以把它保存到文件里，也可以保存到数据库中。这样，当用户下次访问相同页面时，还可以读取到以前的绘图记录。

为了给画图程序增加保存功能，我们使用数据URL的方案。要取得当前数据的URL，必须通过画布对象调用`toDataURL()`方法：

```
var url = canvas.toDataURL();
```

在调用`toDataURL()`方法时如果不提供参数，得到的将是一个PNG图片。如果你想要其他格式的图片，可以传入相应的MIME类型：

```
var url = canvas.toDataURL("image/jpeg");
```

不过，假如浏览器不支持你想要的格式，它仍然会发给你一个PNG文件，一个转换后的长长的字符串。

8

数据URL到底是什么？从技术角度讲，数据URL就是一个以`data:image/png;base64`开头的base-64编码的字符串。这个字符串很长，不过不要紧，因为数据URL是要给计算机程序（如浏览器）看的。以下就是当前画布图片的数据URL：


```
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAfQAAAEsCAYAAAA1uOHIAAAAAXNSR
0IArs4c6QAAAARnQU1BAACxjwv8YQUAACqRSURBVHhe7Z1bkB1Hecdn5uxFFzA2FWOnsEEGiiew
nZgKsrWLrZXMRU9JgZQKhoSHVK...gAAEIQAACEIBAiAT+HxAYpeqDfKieAAAAAElFTkSuQmCC
```


为了节省版面，这里代码的中间省略了大量字符（注意省略号）。如果一点不落地都放在这儿，恐怕得占用本书这样的5页篇幅。

> **注意** Base-64编码是一种将图像数据转换成长字符串的编码方法，长字符串由字符、数字及少量特殊字符组成。由于编码后的字符串不包含标点符号和所有专用扩展字符，所以结果可以安全地用在网页中（例如，作为隐藏输入字段的`value`或`<img>`元素的`src`属性值）。

总之，很容易把画布转换为数据URL形式的图像数据。但有了数据URL之后，又能用它来做什么呢？一种处理方式是将它发送给Web服务器长期保存。这里有一篇文章，讲解了只用少量PHP脚本即可实现上述目的的方法：http://tinyurl.com/5uud9ob。

如果你只想把数据保存在客户端，那方法并不太多。有些浏览器支持直接访问数据URL，也就是可以使用下面这样的代码直接打开图像：

```
window.location = canvas.toDataURL();
```

而更可靠的方法则是把数据URL交给一个`<img>`元素。下面就是我们画图程序的处理代码（见图8-12）：

```
function saveCanvas() {
  //找到<img>元素
  var imageCopy = document.getElementById("savedImageCopy");

  //在图像中显示画布数据
  imageCopy.src = canvas.toDataURL();

  //显示包含<img>元素的<div>，以便把图像显示出来
  var imageContainer = document.getElementById("savedCopyContainer");
  imageContainer.style.display = "block";
}
```

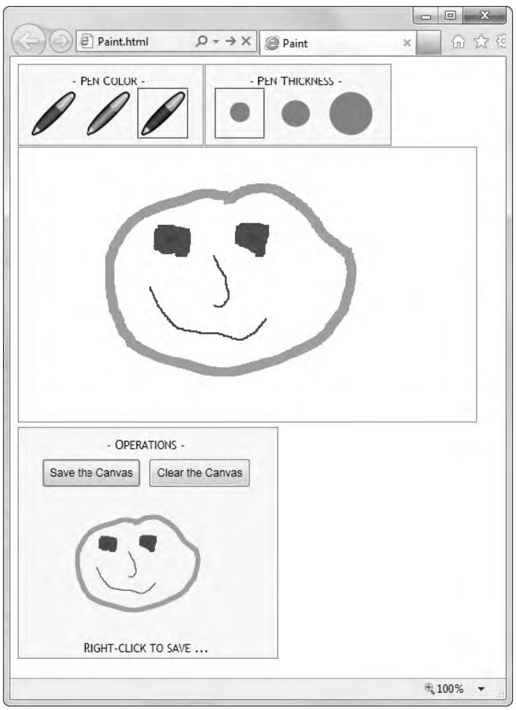


图8-12：在此，我们使用数据URL把画布中的信息传递给了<img>元素。我们把<img>元素的尺寸设置得比较小，以区别于画布。如果想把图像保存为.png文件，只要在图像上单击右键，选择“图片另存为”即可——与保存网页中的其他图像一样

以上代码并没有真的“保存”图像数据，因为图像不会长期保存（比如保存为一个文件）。然而，对于显示在网页中的图像，只要简单操作两下就可以保存下来了。这个大家都知道，在图像上单击右键，选择“图片另存为”就好了。虽然这样不如下载文件或弹出“保存”对话框方便，但却是在所有浏览器中都能可靠使用的唯一一个客户端方案。

注意 Firefox内置支持把画布内容保存为文件。只要在<canvas>元素上（而不是下面的小图像上）单击右键，然后选择“图片另存为”。而Chrome、IE等浏览器则没有提供此功能。

注意 如果你是在本地计算机硬盘上运行自己的试验页面，那么数据URL功能会失效——<canvas>的其他几项功能也是如此。为了避免出现这个问题，需要把试验页面上传到Web服务器后再打开。

基于Canvas的绘图程序

当大家开始学习<canvas>时，首先想到的就是用它来开发绘图程序。因此，在谷歌里一搜，很容易就能找到一些这样的绘图程序，有的还支持很多高级功能。以下是两个例子。

- iPaint（http://tinyurl.com/js-ipaint）。这个简单的程序看起来就像Web版的微软画图。但是，它至少添加了一个新功能——可以选择并维护所绘图像中的对象。
- Sketchpad（http://mugtug.com/sketchpad）。这个装扮得美轮美奂的绘图程序支持了一些先进的图案功能，比如图形管理、选取框、纹理，甚至是动画图。

8.3 浏览器对 Canvas 的支持情况

关于Canvas，我们已经介绍了很多东西了。现在该面对现实，回答那个关系到每个HTML5新功能的问题了——什么时候可以放心使用？

我们很幸运，Canvas是目前得到较好支持的HTML5功能。主流浏览器的所有最新版本都支持它。当然，浏览器版本越高，支持得就越好。而且，新版本浏览器还进一步提高了绘图速度，修复了一些偶尔出现的小问题。

除了IE9之前的版本之外，很少有人会使用不支持<canvas>的旧浏览器。而这正是今天Canvas用户最关心的问题：怎样才能既在网页中使用<canvas>，又能保证不把IE7和IE8这两款浏览器排除在外？

与许多其他HTML5功能一样，为了保证兼容性，我们有两个选择。第一个选择是检测浏览器是否支持新功能，不支持则提供后备内容。第二个选择是利用第三方工具来模仿HTML5的<canvas>，做到同一个页面也能在旧版本浏览器中运行。具体到<canvas>，第二个选择是首选，稍后我们会介绍原因。

8.3.1 用ExplorerCanvas兼容Canvas

为了让过时的老版本IE支持类似Canvas的功能，出现了很多靠谱的方案。较早的一个是ExplorerCanvas库（即excanvas），由谷歌工程师、JavaScript天才Erik Arvidsson开发。这个库能在IE7和IE8中模仿HTML5 Canvas，完全使用JavaScript，还有目前已经不太流行的VML（Vector Markup Language，矢量标记语言）技术。

注意 VML是一种针对在HTML文档中使用标记创建线框图及插图的标记语言规范。目前，已经被与之相似但支持情况更好的标准SVG（Scalable Vector Graphics，可伸缩矢量图）所取代，SVG正开始得到浏览器的广泛支持。VML在微软的某些产品（比如Microsoft Office和IE）中仍然可以使用。因此，虽然其他浏览器支持不好，但在IE浏览器中用它来模仿Canvas仍然是最好的方案。

ExplorerCanvas的下载地址是http://code.google.com/p/explorercanvas。下载后，把excanvas.js文件复制到网页所在目录，然后在网页中引用它就可以了：

```
<head>
  <title>...</title>
  <!--[if lt IE 9]>
    <script src="excanvas.js"></script>
  <![endif]-->
  ...
<head>
```

这里使用了IE的条件注释。只有早于IE9的版本才会使用它，IE9及非IE浏览器会忽略这个脚本。

此后，再使用<canvas>基本上就没有后顾之忧了。实际上，引入这个脚本之后，我们前面开发的画图程序（见8.2节）在旧版本的IE中也可以运行。

注意 如果你想在Canvas上绘制文本（9.1.3节介绍），那还需要第二个JavaScript库的支持，那就是Canvas-text，它与ExplorerCanvas可以协同工作。Canvas-text的下载地址为http://code.google.com/p/canvas-text/。

8.3.2 用FlashCanvas兼容Canvas

当然，ExplorerCanvas也不是完美无缺的。如果你想使用高级的Canvas绘图功能，可能就会引发一些错误。ExplorerCanvas不支持一些主要功能（应该说，在编写本书时还不支持的功能）包括放射渐变、阴影、剪切区域、原始像素处理和数据URL。虽然有可能ExplorerCanvas将来会更新并加上这些功能，但这种可能性不大——ExplorerCanvas当前的版本已经是几年前的了，而且代码也有段时间没更新了。

如果你的要求很高（比如，也许你想创建复杂的动画或横向卷轴游戏），ExplorerCanvas的性能恐怕无法满足需要。在这种情况下，建议你考虑其他基于高性能浏览器插件（如Silverlight或Flash）的JavaScript库。GitHub上有一个页面，地址是http://tinyurl.com/polyfills，其中列出了几乎所有可供选择的JavaScript补丁库。其中哪个最好？现在就可以推荐一个免费的库：FlashCanvas，下载地址是http://flashcanvas.net/purchase[1]。与ExplorerCanvas类似，要使用FlashCanvas，只需一行脚本。但与ExplorerCanvas不同的是，FlashCanvas依赖Flash插件，而不使用VML。

FlashCanvas还有一个更完善的专业版FlashCanvas Pro。它添加了一些额外功能的支持，比如全部的合成操作（参见8.1.6节）和阴影（参见9.2.1节）。

FlashCanvas Pro对非商业使用是免费的（可以在http://flashcanvas.net/download下载）。如果是营利性质的机构或个人，需要花一笔小钱（目前是31美元），购买链接为http://flashcanvas.net/purchase。至于ExplorerCanvas、FlashCanvas与FlashCanvas Pro对Canvas的支持情况对比，可以参考这个页面：http://flashcanvas. net/docs/canvas-api。

和ExplorerCanvas一样，FlashCanvas项目最近也不太活跃了。但是，以它目前的状态，仍然是旧版浏览器兼容<canvas>的可靠选择。如果想要做一个宏大的基于<canvas>的网站，比如实时游戏，那就需要测试FlashCanvas，看它是否支持所需的功能。

注意 使用FlashCanvas开发基于Canvas的应用，能够在如今几乎所有的浏览器中得到最好的支持。不仅能通过Flash获得版本老一些的IE的支持，而且在不支持Flash的iPad、iPhone等移动设备上，同样能通过HTML5获得原生支持。

8.3.3 Canvas后备及功能检测

8

扩展网页对Canvas支持的最流行方案是使用ExplorerCanvas和FlashCanvas。但它们并不是唯一的方案。

与上一章介绍的<audio>和<video>元素一样，<canvas>元素本身也支持后备内容。比如，下列代码表示可以在浏览器支持的情况下使用<canvas>，而在不支持的情况下显示图像：

```
<canvas id="logoCreator" width="500" height="300">
  <p>The canvas isn't supported on your computer, so you can't use our
  dynamic logo creator.</p>
  <img src="logo.png" alt="Standard Company Logo">
</canvas>
```

这种方案当然是聊胜于无。多数时候，我们会利用<canvas>绘制一些动态图像，或者创建一些基于图形的交互应用，在这种情况下只显示一幅静态图像，显然于事无补。为此，更好的办法是在<canvas>元素中嵌入Flash应用。这个办法对已经有了Flash版，又想迁移到<canvas>以适应未来发展的应用非常合适。这样，既有满足老版本IE浏览器的Flash应用，又可以让其他人安心使用

注1：原书给出的下载地址是http://code. google. Com/p/flashcanvas，但该网站可能已经不存在。（编者注）

无插件的<canvas>版。

如果你使用Modernizr（参见1.6.7节），还可以在JavaScript代码中检测浏览器是否支持<canvas>。为此，只要检测Modernizr.canvas属性即可。而要检测浏览器是否支持文本绘制功能（后来才添加的Canvas绘图功能），可以检测Modernizr.canvastext属性。如果你不想检测浏览器对<canvas>的支持情况，可以选择任何自己喜欢的JavaScript补丁库。

无障碍的Canvas

是否可以让残疾人无障碍地访问Canavs？

语义元素是HTML5的重头戏，而本书之前几章也一直在强调**无障碍性**，即在设计页面时，就要考虑为残疾人使用的辅助上网工具提供信息，以方便残疾人使用你的网站。一路下来，突然冒出了另一个HTML5中头等重要的元素，但它却没有语义，也不支持无障碍访问。

HTML5的制定者正在积极考虑弥补这个问题。然而，什么方案最好，目前还没有定论。有人建议为辅助上网工具单独生成一个文档，该文档完全镜像Canvas的内容。可问题在于，这个镜像文档终归还要让网页作者生成，这个“影子”文档能否与可见内容同步，就会因作者而异。如果创建镜像文档很麻烦，那么想偷懒的人，或者手头活太多的人可能就会对其敷衍了事、推卸责任。

另一种意见是扩展图像地图（已有的一种HTML功能，可以把一张图像切分成多块可单击的区域），以便将其作为独立的一层放在Canvas上面。由于图像地图本质上就是一组链接，所以可以在其中保存重要信息，供辅助上网工具读取并传达给残疾用户。

目前来看，对这两种意见我们只能姑妄听之，因为都还停留在讨论阶段。在此期间，你尽管使用Canvas去实现各种图形开发任务，比如大型电子游戏（电子游戏实际上并不能做到无障碍），或者数据可视化（只要相关数据有文本格式，就可以做到无障碍）。但是，不能把Canvas当做一个通用万能的页面设计元素。换句话说，如果你只想用它来创建一个新奇的标题或者网站导致菜单，我劝你还是省省吧，最好现在就打消这个念头。

第 9 章

高级Canvas技术：交互性和动画

Canvas功能十分庞大，而且还在逐步发展。上一章，我们学习了如何绘制直线，乃至使用为数不多的JavaScript代码开发一个还像那么回事儿的画图程序。可是，Canvas本身的功能还远不止这些。使用它不仅能显示动态图片、开发画图工具，还可以播放动画、在像素级别上处理图像，甚至基于它创建交互游戏。这一章，我们介绍上述所有功能的实际应用。

首先，我们从绘图上下文支持的绘制图像和文本的不同方法讲起，然后讨论为图像添加阴影、使用图案和渐变填充。最后，学习为Canvas添加交互功能，以及通过它实现动画效果。最关键的是，实现这些功能只要编写基本的JavaScript代码即可，当然还要有你的创意。

注意 本章前半部分重点分析小段的绘图代码。你可以把这些代码直接放到自己的网页中，但别忘了在页面中添加相应的<canvas>元素，以创建绘图上下文（参见8.1节）。本章后半部分讲解更复杂的任务，书中会给出相关示例的大部分（或全部）绘图代码，但不会给出每个页面的标记代码。如果你想自己动手试验这些例子，请访问本书试验站点 http://prosetech.com/html5。

9.1　高级 Canvas 绘图

使用Canvas可以绘制你能想到的任何图形，无论是线条、三角形，还是着色讲究的人物素描。但绘图任务越复杂，代码自然也越复杂。很多时候，要得到精细的最终结果，靠手工编写每一行代码是不现实的。

好在我们有的选择。绘图上下文不只是能绘制直线和曲线，它还支持各种方法，让我们能直接绘制已有的图片、文本、图案，甚至视频帧。接下来的几节就介绍如何使用这些方法，从而在画布中生成更丰富的内容。

9.1.1　绘制图像

大家都见过使用卫星图片构建的网页地图吧，其中的地图切片都是下载后又拼合到一起的。这个典型的例子说明，我们可以利用已有图片，将它们组织成最终作品。

绘图上下文提供了drawImage()方法，用于在画布上绘制图片。使用这个方法很简单，调用它的时候传入相应的图片对象及起点坐标即可：

```
context.drawImage(img, 10, 10);
```

虽然，在调用drawImage()之前，需要准备好图片对象。HTML5为此提供了三个方案。首先，可以使用createImageData()方法一个像素一个像素地创建图像。这种方法很麻烦，也没有效率（但9.5.3节仍然会介绍像素级操作）。

其次，是使用网页中已有的<img>元素。比如，假设网页中存在如下标记：

```
<img id="arrow_left" src="arrow_left.png">
```

那么，使用以下代码就可以把该图片复制到画布上：

```
var img = document.getElementById("arrow_left");
context.drawImage(img, 10, 10);
```

第三种方案是在代码中创建一个图片对象，然后把一个外部图片加载进来。但这个方案有一个缺点，即必须先等待图片加载完毕，然后才能把图片对象传递给drawImage()方法使用。为此，需要等待图片对象的onLoad事件发生，然后再处理图片。

为了理解这个过程，最好看一个例子。假设我们有一张名为maze.png的图片，你想把它显示在画布上。理论上讲，应该通过如下几步实现：

```
//创建图片对象
var img = new Image();

//加载图片文件
img.src = "maze.png";
//绘制图片（可能会因为图片尚未加载完而导致失败）
context.drawImage(img, 0, 0);
```

以上代码的问题是设置图片对象的src属性后只是开始加载外部图片，但代码没有等到加载完成就立即执行绘图操作。对此，正确的方式是像下面这样：

```
//创建图片对象
var img = new Image();

//添加onload事件处理程序
//告诉浏览器在图片加载完成后该做什么
img.onload = function() {
  context.drawImage(img, 0, 0);
};

//加载图片文件
img.src = "maze.png";
```

乍一看，这有点违反直觉。因为代码的顺序与执行顺序并不一致。对这个例子而言，context.drawImage()实际上会后执行，也就是会在设置img.src属性的代码执行后才执行。

有了图片，就可以实现很多新奇的功能。比如，可以用它们来装饰自己的线条图作品，可以直接绘制图片而节省手工绘制时间。如果是在游戏里，可以使用图片来表示物体和人物，把它们

分别摆放在画布的不同位置上。而在画图程序里使用图片代替线段，就可以画出“纹理化”的线条来。本章会介绍一些使用图片绘图的实用技术。

我的图片变形了

在绘制图片时，如果你发现原来的图片不知为什么被拉长了、压扁了，总之变形了，那幕后黑手很可能是样式表规则。

为画布指定宽度和高度的最佳方案，就是在HTML标记中使用width和height属性。可能有人觉得像下面这样使用标记更简洁：

```
<canvas></canvas>
```

因为可以通过样式表规则来控制画布的大小，比如：

```
canvas {
  height: 300px;
  width: 500px;
}
```

但这个方案行不通！问题在于，CSS的width和height属性与画布的width和height属性并不是一回事儿。假如你真的这么做了，那画布会取得其默认尺寸（300像素×150像素）。然后，CSS的width和height属性又会把画布拉伸或压缩到它设置的大小。与此同时，画布中的内容也会随之变形。结果，在通过画布显示图片时，图片也会被压扁，这显示会降低图片的吸引力。

为避免这个问题，请一定要在HTML标记中为画布指定宽度和高度。如果你想在某个条件下改变画布的大小，可以使用JavaScript代码来修改<canvas>元素的宽和高。

9.1.2 裁剪、切割和伸缩图片

可以给drawImage()函数传递一些可选的参数，从而影响在画布上绘制图片的方式。首先，如果想改变图片的大小，可以添加宽度和高度，例如：

```
context.drawImage(img, 10, 10, 30, 30);
```

这就相当于为图片准备了一个30像素×30像素的方框，其左上角在画布上的坐标为(10,10)。假设图片实际上是60像素×60像素，则执行上面的代码会把图片的宽度和高度都缩小一半，最终在画布上呈现的大小只有原来的1/4。

如果想裁剪掉一部分图片，可以再为drawImage()函数传入4个参数，这个4个参数从图片对象参数后面开始。之所以传入4个参数，正是为了定义从原始图片的什么位置，裁剪多大的图片，每个参数的含义如下所示：

```
context.drawImage(img, source_x, source_y,
 source_width, source_height, x, y, width, height);
```

最后4个参数与上一个例子中的相同，它们定义被裁剪后的图片在画布上的位置和大小。

比如有一张200像素×200像素的图片，但我们只想在画布上绘制它的上半部分。为此，就要创建一个200像素×100像素的矩形框，从原始图片的(0,0)位置开始裁剪，得到图片的上半部分。

然后，把裁剪后的结果绘制到画布上，起点为(75,25)。用代码表示就是：

```
context.drawImage(img, 0, 0, 200, 100, 75, 25, 200, 100);
```

图9-1演示了这个例子的结果。



图9-1：左：原始图片。右：把裁剪结果绘制到画布上

如果你想要实现的效果更多，比如先扭曲和旋转图片，然后再绘制，那drawImage()方法就不够用了。不过，可以使用8.1.4节学习的变换来修改绘制所有内容的方式。

绘制视频帧

我们知道，drawImage()方法的第一个参数是要绘制的图片。如前所述，这个图片可以是临时创建的图片对象，也可以是页面某个地方已经存在的<img>元素。

但这些并非HTML5为drawImage()定义的全部功能。实际上，除了绘制图片，还可以绘制整个<canvas>元素（不是当前的这个）。另外，还可以绘制目前正在播放的<video>元素，且无需额外的工作：

```
var  video  =
 document.getElementById("videoPlayer");

context.drawImage(video,  0,  0,
 video.clientWidth, video.clientWidth);
```

以上代码运行后，会捕获代码运行瞬间正在播放的视频中的一帧画面，然后把该画面绘制到画布上。

这就为实现其他很多有趣的效果提供了可能。例如，可以利用一个计时器来不断捕获播放中的视频，然后不断将新画面绘制到画布上。假如整个过程足够快，则复制的画面在画布上连续播放，就会成为另一个视频播放器。

发挥一点想象，比如可以在绘制画面之前，对其进行一些修改。比如，可以放大或缩小画面，或者取得其中的像素数据，然后应用Photoshop滤镜般的效果。要了解这方面的实际示例，可以参考这篇文章：http://html5doctor.com/video-canvas-magic。这篇文章里介绍了在画布中播放黑白画面的实现过程，方法就是从现有视频中实时取得画面截图，然后把每个彩色像素转换成黑白像素，最后再绘制到画布上。

9.1.3 绘制文本

除了直线和曲线，你一定还想在画布上绘制文本，但你肯定不愿意自己通过绘制线条来形成文本。HTML5规范也没有认为你愿意。为此，我们就有了另外两个绘图上下文方法支持绘制文本。

首先，在绘制文本之前要设置绘图上下文的font属性。这个属性的值是一个字符串，与设置CSS的font属性时使用的“多合一”的值相同。最简单的情况，也要设置字体大小（像素）和字体名称，比如：

```
context.font = "20px Arial";
```

如果不能确定用户的浏览器支持哪种字段，可以多列出几种来：

```
context.font = "20px Verdana,sans-serif";
```

此外，还可以为字体应用加粗效果，不过要把它放在字符串的开头：

```
context.font = "bold 20px Arial";
```

而在CSS3的支持下，甚至还可以使用新颖的网络字体。这要涉及使用样式表来注册字体，相关内容将在6.4节介绍。

设置好字体后，就可以调用fillText()方法绘制文本内容了。以下示例代码将把文本内容的左上角放在画布的(10,10)坐标点处：

```
context.textBaseline = "top";
context.fillStyle = "black";
context.fillText("I'm stuck in a canvas. Someone let me out!", 10, 10);
```

可以把文本内容绘制到任何地方，但每次却只能绘制一行。如果要绘制多行文本，那只能多次调用fillText()方法。

提示 如果要把一个完整的段落折成多行文本，可以设计自己的**单词折行算法**。基本的思路就是把句子拆成单词，然后通过绘图上下文的measureText()方法获悉每行可以放下多少个单词。这个工作确实很烦琐，不过可以参考这里的示例代码，也许你能从中得到一些启发：http://tinyurl.com/6ec7hld。

除了fillText()方法，还有另一个绘制文本的方法，即strokeText()。这个方法用于绘制文本的轮廓，轮廓的颜色取自strokeStyle属性，而轮廓的宽度取自lineWidth属性。下面是一

个例子：

```
context.font = "bold 40px Verdana,sans-serif";
context.lineWidth = "1";
context.strokeStyle = "red";
context.strokeText("I'm an OUTLINE", 20, 50);
```

使用strokeText()时，文本的中部是空白的。当然，如果你想得到加了彩色描边的文本，可以先调用fillText()绘制实心文本，然后调用strokeText()绘制文本的轮廓。图9-2展示了这两个方法绘制的文本。

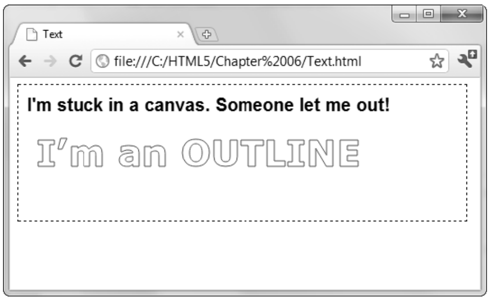


图9-2：在画布上绘制实心文本和空心文本都很简单

提示 与绘制线条和图片相比，绘制文本的速度稍慢一些。如果你想创建静态、不变的图像（如数据图表），这个速度不是问题。但是，如果你想创建交互、动态的应用，那么绘制文本的速度可能就会影响性能。至于优化速度的手段，可能就是事先把文本保存为图片，然后再使用drawImage()把图片绘制到画布上。

9.2 阴影与填充

现在，我们在画布上绘制线条和填充图形时，使用的都是实色。这当然没有问题，不过要是我告诉大家Canvas还有一些新奇的绘图功能，有创意的设计师一定会兴高采烈。比如，可以在画布上为图形绘制模糊的阴影，可以用小图案来填充图形。当然，最令人拍手叫绝的还是渐变功能，把多种颜色组合起来，能够形成千变万化的模式。

接下来的几节我们就来介绍这些新功能，实际上只要简单地设置绘图上下文的另外一些属性就好了。

9.2.1 添加阴影

为绘制的任何内容添加阴影是Canvas最便利的功能。图9-3展示了几种阴影的示例。



图9-3：阴影与形状、图片和文本一样。特别是给带透明背景的图片加阴影时，阴影的形状会随不透明部分的形状变化，比如图中右上角的五角星，其阴影的形状也是相同的五角形，而不是方形。（在编写本书时，还只有IE和Firefox支持这个功能。）阴影与文本也是相辅相成，而且设置不同，阴影的效果也不一样

本质上，可以把阴影看成原来绘制内容（直线、形状、图片或文本）的模糊版。控制阴影的外观，需要使用绘图上下文的几个属性，如表9-1所示。

表9-1　阴影相关的属性

属　　性	说　　明
shadowColor	设置阴影颜色。可以把阴影设置为黑色或彩色，但中性灰还是最佳选择。另外一种技术是使用半透明的颜色（参见6.1.5节），以便下方内容可以若隐若现。在不需要阴影的时候，可以把shadowColor设置为完全透明
shadowBlur	设置阴影的模糊程度。值为0表示锐利的阴影，结果会生成原始形状的一个轮廓鲜明的副本。相对来说，值为20的时候已经比较模糊了，不过当然还可设置更大的值。一般来说，这个值不小于3才会达到最佳效果
shadowOffsetX shadowOffsetY	设置阴影相对于内容的位置。例如，把这两个属性都设置为5，会导致阴影被绘制到原图形向右和右下各5像素的位置。使用负值可以把阴影移动到其他位置（左和上）

9

以下是创建图9-3中所示各种阴影的代码：

```
//绘制矩形阴影
context.rect(20, 20, 200, 100);
context.fillStyle = "#8ED6FF";
context.shadowColor = "#bbbbbb";
context.shadowBlur = 20;
context.shadowOffsetX = 15;
context.shadowOffsetY = 15;
context.fill();

//绘制星形阴影
context.shadowOffsetX = 10;
context.shadowOffsetY = 10;
```

```
context.shadowBlur = 4;
img = document.getElementById("star");
context.drawImage(img, 250, 30);

context.textBaseline = "top";
context.font = "bold 20px Arial";

// 绘制三行文本的阴影
context.shadowBlur = 3;
context.shadowOffsetX = 2;
context.shadowOffsetY = 2;
context.fillStyle = "steelblue";
context.fillText("This is a subtle, slightly old-fashioned shadow.", 10, 175);

context.shadowBlur = 5;
context.shadowOffsetX = 20;
context.shadowOffsetY = 20;
context.fillStyle = "green";
context.fillText("This is a distant shadow...", 10, 225);

context.shadowBlur = 15;
context.shadowOffsetX = 0;
context.shadowOffsetY = 0;
context.shadowColor = "black";
context.fillStyle = "white";
context.fillText("This shadow isn't offset. It creates a halo effect.", 10,
300);
```

9.2.2 填充图案

说到填充，我们到目前为止用到的都是实色或部分透明的颜色。除此之外，还可以使用图案和渐变。图案和渐变可以让平淡无奇的图形一下子变得活泼起来。这两种填充方式很简单，只要两步。首先，创建要填充的内容。然后，将其添加到fillStyle属性（有时候需要使用strokeStyle属性）。

要实现用图案填充，首先要选择一张小图片，而且要能够前后左右拼接在一起覆盖一块大区域（参见图9-4）。当然，需要利用前面介绍的技术把这张图片加载到图片对象中，比如在页面中放一个隐藏的<img>元素（参见9.1节），或者使用代码创建图片对象，把外部图片加载进来，然后处理图片对象的onLoad事件（参见9.1.1节）。在此，我们使用第一种方法：

```
var img = document.getElementById("brickTile");
```

有了图片对象后，就可以利用绘图上下文的createPattern()方法创建一个图案对象。此时，可以选择图案是水平（repeat-x）、垂直（repeat-y），还是在两个方向（repeat）重复：

```
var pattern = context.createPattern(img, "repeat");
```

最后是使用图案对象设置fillStyle或strokeStyle属性：

```
context.fillStyle = pattern;
context.rect(0, 0, canvas.width, canvas.height);
context.fill();
```

这样，就创造出了用小幅图片填充画布的效果，如图9-4所示。

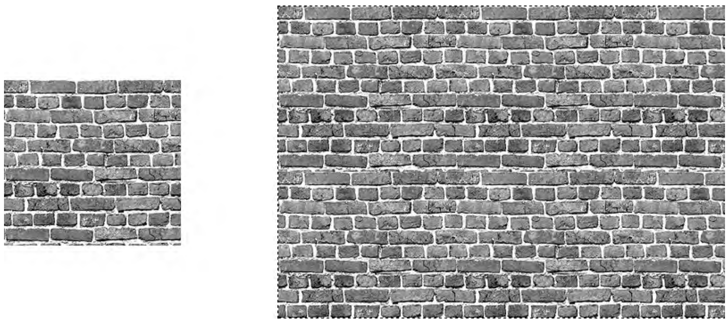


图9-4：左：作为图案的图片。右：通过拼接图案形成的整体效果

9.2.3 填充渐变

第二种填充形式是渐变，也就是混合在一起的两种或多种颜色。Canvas支持线性渐变和放射性渐变，图9-5展示了这两种渐变形式。

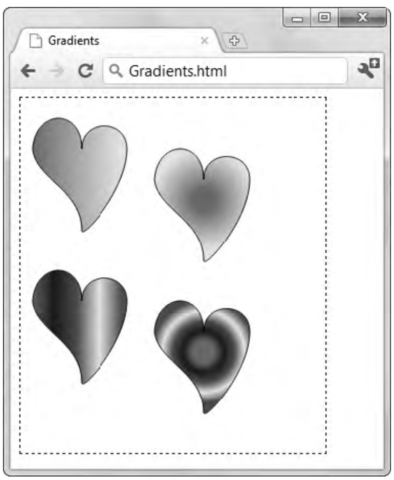


图9-5：线性渐变（左上）是在一个方向上混合色彩。放射性渐变（右上）是从一点向四周混合色彩。这两种渐变形式都支持多种颜色混合，因此使用线性渐变可以创造出色谱效果（左下），使用放射性渐变可以创造出同心圆扩散的效果（右下）

提示　如果读者是在黑白纸质书上看这个图中的渐变效果，建议你访问http://prosetech.com/html5，看看真实网页中丰富的色彩，估计会震撼你一小下。（网页中的绘图代码也包含绘制心形的逻辑，每个心形都是通过把4条贝塞尔曲线连接成一段路径做出来的。）

估计读者也猜到了，使用渐变填充的第一步是创建渐变对象。绘图上下文为此提供了两个方法：createLinearGradient()和createRadialGradient()。这两个方法的用法大致相同，即它们接收一组坐标，表示不同颜色的起点。

理解渐变的最简单方式就是看一个例子。以下代码创建的是图9-5左上角心形的渐变：

```
//创建一个从(10,0)到(100,0)的渐变
var gradient = context.createLinearGradient(10, 0, 100, 0);

//添加两种颜色
gradient.addColorStop(0, "magenta");
gradient.addColorStop(1, "yellow");

//调用另一个函数绘制心形
drawHeart(60, 50);

//填充心形
context.fillStyle = gradient;
context.fill();
context.stroke();
```

这里是创建线性渐变，因此我们给createLinearGradient()传入两个坐标点，分别表示渐变的起点和终点。起点和终点构成了颜色逐渐过渡的区间。

起点到终点的渐变线很重要，因为它决定了渐变的最终效果（参见图9-6）。例如一个从品红过渡到黄色的线性渐变，这个渐变可以在几个像素的距离上完成，也可以跨越整个画布的宽度。而且，渐变可以是从左到右，也可以是从上到下，甚至发生在两个任意点之间（渐变线的角度可以任意变化）。总之，渐变线决定了这一切。

提示 可以把渐变想象成位于画布下方的彩色图样。在创建渐变时，你是在创建这个彩色但却隐藏的图样。而在填充图形时，你会在画布上按照图形的形状抠出一个洞，从而让下面那部分图样显示出来。实际的效果（在画布上呈现的结果），取决于渐变的设置和形状的大小及位置。

对这个例子而言，渐变线的起点和终点分别是(10,0)和(100,0)。这两个点决定以下重要信息。

- **渐变是水平的**。也就是说，渐变的颜色将从左到右混合。之所以知道渐变是水平的，是因为这两个点的*y*轴坐标相等。如果你想创建从上到下的渐变，可以把起点和终点设置为(0,10)和(0,100)。类似地，对角线方向的渐变（从左上到右下），可以使用(10,10)和(100,100)。
- **实际的混合颜色区宽度为90个像素（*x*坐标从10到100）**。在这个例子中，心形比渐变范围稍小一些，因此可以在心形里看到大部分渐变。
- **超过渐变范围的颜色会变成实色**。因此，如果把心形设置得更宽，就会看到更多品红（左）和黄色（右）。

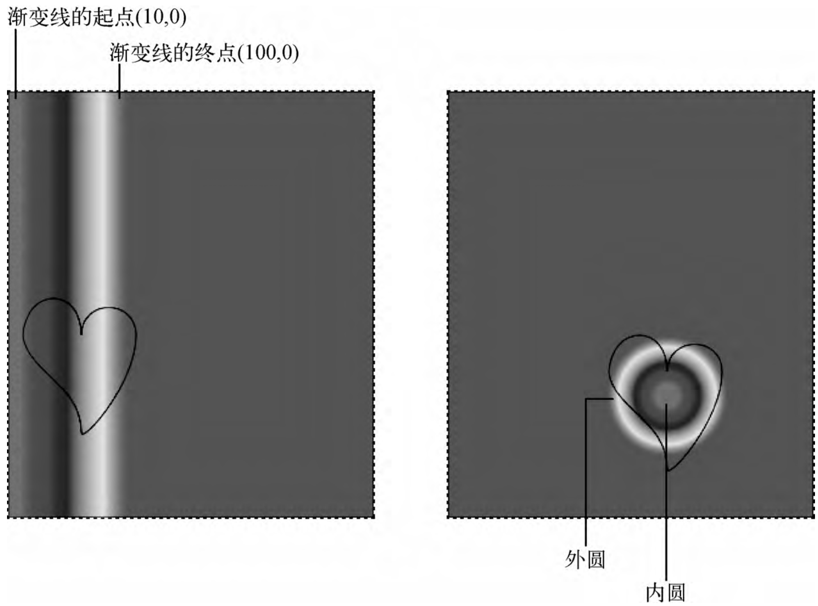


图9-6：左：以下是为图9-5左下角的心形生成的渐变。用这个渐变填充心形后，只能看到整个渐变的一部分。右：为图9-5右下角的心形生成的放射性渐变，同样也只是整个渐变的一部分

提示 一般来说，我们创建的渐变只要恰好比要填充的图形大一点即可，正如这个例子所示。当然，也还有其他可能。比如，要是你想利用一个渐变的不同部分填充几个图形，可能就需要创建一个画布那么宽的渐变。

确定了渐变线的宽度和角度之后，接下来就该实际地设置构成渐变的颜色了。要设置渐变颜色，需要使用渐变对象的addColorStop()方法。每次调用这个方法，都需要提供一个0～1的偏移值和一个颜色值（颜色名）。其中，偏移值决定颜色在渐变中的位置：0表示位于渐变的起点，1表示位置渐变的终点。改变这两个值（比如，分别改为0.2和0.8），就会压缩渐变的范围，让两端显示出更多的实色。

在创建双色渐变时，最好将0和1分别作为两种颜色的偏移值。而在创建多种颜色构成的渐变时，可以通过选择不同的偏移值来加宽某种颜色区的宽度，或者缩小某种颜色的范围。图9-5中左下角心形渐变的偏移值是平均分布的，即每种颜色的范围都一样宽：

```
var gradient = context.createLinearGradient(10, 0, 100, 0);
gradient.addColorStop("0", "magenta");
gradient.addColorStop(".25", "blue");
gradient.addColorStop(".50", "green");
gradient.addColorStop(".75", "yellow");
gradient.addColorStop("1.0", "red");

drawHeart(60, 200);
context.fillStyle = gradient;
```

```
context.fill();
context.stroke();
```

注意 如果此时此刻你感觉天旋地转，别慌。实际上，你不用理解渐变是怎么来的。你只要会调整偏移值就够了，不行就反复调用，直至得到满意的渐变效果为止。

创建放射性渐变与创建线性渐变类似。只不过，这次不是指定两个点，而是要指定两个圆。这是因为放射性渐变就是颜色从一个小圆过渡到一个更大的、包含它的圆。要定义圆，需要提供圆心坐标和半径。

在图9-5右上角那个放射性渐变的例子中，渐变的起点是在心形内部，坐标为(180,100)。内部颜色由一个半径为10像素的圆表示，外部颜色由一个半径为50像素的圆表示。同样，在小圆内部或者大圆外部（即超出两个圆范围之外的地方）会显示实色，因此该放射性渐变的中心是品红色，而外围是实心黄色。

以下是创建这个双色放射性渐变的代码：

```
var gradient = context.createRadialGradient(180, 100, 10, 180, 100, 50);
gradient.addColorStop(0, "magenta");
gradient.addColorStop(1, "yellow");

drawHeart(180, 80);
context.fillStyle = gradient;
context.fill();
context.stroke();
```

注意 把两个圆设置为同心圆是最常见的做法。不过，当然可以给内圆和外圆设置不同的圆心，而这样可以实现拉伸、压缩或其他颜色变形效果。

在这个例子的基础上，我们可以创建图9-5右下角那个多色放射性渐变效果。只要把两个圆的圆心坐标平移到那个心形的内部，然后再（使用渐变对象的addColorStop()方法）加上不同的色标（与创建多色线性渐变时使用的色标相同）即可：

```
var gradient = context.createRadialGradient(180, 250, 10, 180, 250, 50);
gradient.addColorStop("0","magenta");
gradient.addColorStop(".25","blue");
gradient.addColorStop(".50","green");
gradient.addColorStop(".75","yellow");
gradient.addColorStop("1.0","red");

drawHeart(180, 230);
context.fillStyle = gradient;
context.fill();
context.stroke();
```

好了，以你现在掌握的技术，要创建出光怪陆离的图案已经不成问题了。

9.2.4 综合示例：绘制图解

既然你已经历尽千难万险，征服了Canvas绘图更具有挑战性的功能，现在该停下来好好享受一下胜利果实了。下面，我们将介绍一个示例，看看怎么利用Canvas把一堆毫无吸引力的文本和数字，转换成简单而漂亮的图解。

图9-7展示了这个示例的起点状态：由两个页面组成的个性测试，其中点缀着一些图片。用户在第一个页面回答问题，然后点击Get Score转到下一页。第二页根据第一页众所周知的“大五人格理论”得到个性测试的得分（参见后面的附注栏）。

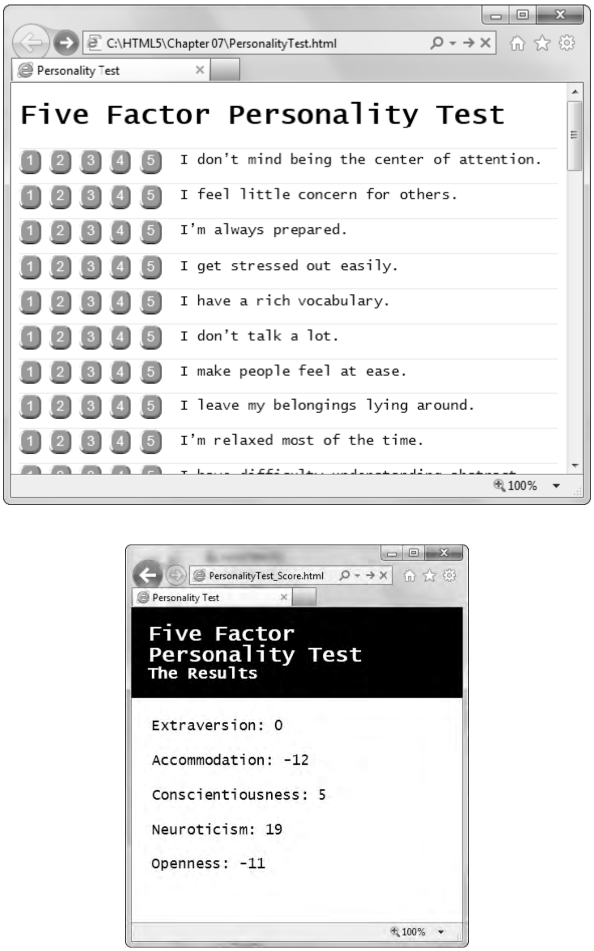


图9-7：点击回答问题（上），然后看看得分（下）。然而，这个测试没有可视化的刻度，一般人很难理解结果分数的具体含义

这个示例的JavaScript代码非常容易理解。在用户单击一个数字按钮时，按钮背景会改变，以反映用户的选择。而用户回答完所有问题后，会有一个简单的算法，把答案传给一组计分公式，从而计算出5个人格因素。如果你想看一下完整的代码，或者想动手试一试，可以访问

http://prosetech.com/html5。

到目前为止，还没有使用HTML5。不过，请大家考虑一下怎么改进这个两页的人格测试示例，比如通过图解形式显示5个人格因素的得分情况。图9-8展示了对这个人格测试的结果页面进行改进之后的结果，即以图解形式显示了得分。

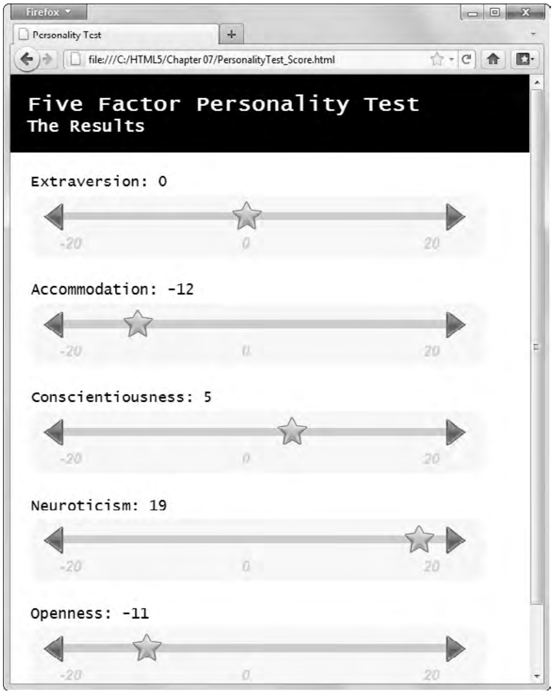


图9-8：这个页面使用了几种不同的绘图方式，绘制了直线、图像和文本。但最关键的地方还是根据测试答案动态绘制的图示

怎样把人格转换成5个数字

大五人格测试根据每个人的五方面人格“因素”来确定性格。这五方面因素是：开放性、责任心、外倾性、亲和力和情绪稳定性。这些因素是研究人员在分析了人们用成千上万个英语形容词描述的性格特征之后提炼出来的。

为了得到五方面信息，心理学家综合运用了基准统计信息、个性调查和计算机。他们希望知道人们会选择哪些形容词，然后据以提炼出最小的性格特征。比如，认为自己**乐于助人**的人，一般会把自己描述为**喜欢社交**和**过集体生活**，因此就可以把这些特征归纳为一个人格因素（心理学家称其为**外倾性**）。经过对近两万个形容词的研究，他们最终归纳出五个最相关的因素。

要了解“大五性格模型”的更多信息，读者可以参考http://tinyurl.com/big-five-p，或*Your Brain: The Missing Manual*（O’Reilly）。

为了显示图解，结果页面中使用了5个Canvas，每个对应一个人格因素。以下是相应的标记：

```
<header>
  <h1>Five Factor Personality Test</h1>
  <p>The Results</p>
</header>

<div class="score">
  <h2 id="headingE">Extraversion: </h2>
  <canvas id="canvasE" height="75" width="550"></canvas>
</div>

<div class="score">
  <h2 id="headingA">Accommodation: </h2>
  <canvas id="canvasA" height="75" width="550"></canvas>
</div>

<div class="score">
  <h2 id="headingC">Conscientiousness: </h2>
  <canvas id="canvasC" height="75" width="550"></canvas>
</div>

<div class="score">
  <h2 id="headingN">Neuroticism: </h2>
  <canvas id="canvasN" height="75" width="550"></canvas>
</div>

<div class="score">
  <h2 id="headingO">Openness: </h2>
  <canvas id="canvasO" height="75" width="550"></canvas>
</div>
```

每个图示都使用了同一个自定义JavaScript函数，plotScore()。这个页面调用了plotScore()函数5次，每次都传入不同的参数。例如，在页面顶部绘制“外倾性”的图示时，传递的参数是最顶部的Canvas元素、分数（从–20～20的值），以及文本标题（“Extraversion”）：

```
window.onload = function() {
  ...
  //取得显示外倾性图示的画布
  var canvasE = document.getElementById("canvasE");

  //将分数添加到对应的标题后面
  //（分数保存在变量extraversion里）
  document.getElementById("headingE").innerHTML += extraversion;

  //在对应的画布中标绘分数
  plotScore(canvasE, extraversion, "Extraversion");
  ...
}
```

下面再看看plotScroe()函数，该函数执行一系列绘图代码，根据前面介绍的知识，读者应该不难理解这些代码。总之，代码中使用了各种绘图上下文的方法，绘制了分数图示的不同部分：

```
function plotScore(canvas, score, title) {
  var context = canvas.getContext("2d");

  //在图示的两端绘制箭头
```

```
    var img = document.getElementById("arrow_left");
    context.drawImage(img, 12, 10);
    img = document.getElementById("arrow_right");
    context.drawImage(img, 498, 10);

    //绘制箭头之间的刻度线
    context.moveTo(39, 25);
    context.lineTo(503, 25);
    context.lineWidth = 10;
    context.strokeStyle = "rgb(174,215,244)";
    context.stroke();

    //把数值写在刻度位置上
    context.fillStyle = context.strokeStyle;
    context.font = "italic bold 18px Arial";
    context.textBaseline = 'top';

    context.fillText("-20", 35, 50);
    context.fillText("0", 255, 50);
    context.fillText("20", 475, 50);

    //绘制星星，显示分数在图示上的位置
    img = document.getElementById("star");
    context.drawImage(img, (score+20)/40*440+35-17, 0);
  }
```

最重要的是最后一行代码，这行代码通过有点不好理解的公式，把星星绘制在正确的位置上：

```
context.drawImage(img, (score+20)/40*440+35-17, 0);
```

这里稍微解释一下。首先是把分数转换为0～100的百分比值。因为分数一般会落在–20～20这个区间内，所以代码第一步要把这个分数转换成0～40的值：

```
score+20
```

而用这个值除以40就可以得到百分比值：

```
(score+20)/40
```

得到百分比值后，接下来需要用它乘以刻度线的长度。0%表示在最左端，100%表示在另外一端，而其他百分比值的结果就是位于两端之间：

```
(score+20)/40*440
```

如果刻度线的*x*坐标是从0到400，这个公式就已经够用了。但实际上，这条线是从画布左边偏右一点绘制的，目的是为了留下一些空间。因此，需要在绘制星星时也偏移相应的像素数：

```
(score+20)/40*440+35
```

可是，这样只把星星的左边放到正确的位置上。而我们实际上是想把星星的中心点放在该位置。为了补偿这个距离，需要再减去星星宽度的一半：

```
(score+20)/40*440+35-17
```

这就是根据分数计算得到的星星的*x*坐标了。

注意 从静态绘图到这个例子中所展示的根据数据来动态绘图，应该说只是向前跨越了一小步。而哪怕是跨越了这一小步之后，你就具备了创建各种数据驱动图表的经验，无论是传统的饼图，还是使用刻度盘和计量仪的信息图。什么，有没有简化工作的工具？有啊，推荐大家使用Canvas图形库，这些库包含写好的JavaScript函数，可以根据你的数据绘制常见的图表。比如，RGraph（http://www.rgraph.net/）和ZingChart（http://www.zingchart.com/）都是不错的选择。

9.3 赋予图形交互能力

Canvas是一种非保留性的绘图界面。换句话说，它不会记录过去执行的绘图操作，而只是保持最终结果——构成图像的彩色像素。

比如，你要在画布中央绘制一个红色的正方形，调用`stroke()`或`fill()`之后，那个正方形仅仅就是包含红色像素的正方形区域。Canvas不会保存这个正方形区域。

这个模型能保证绘图速度，但同时也导致不便为绘制的图形添加交互性。假设你想为图8-11所示的画图程序创建一个更智能的版本，比如不仅支持画线，还支持画矩形。（支持画矩形不难。）而且，不仅支持画矩形，还要支持让用户选择、拖动矩形，以及调整矩形大小、改变颜色，等等。在实现这些功能之前，必须理清几方面思路。首先，怎么知道用户选择了矩形？其次，怎么知道矩形的相关信息，比如坐标、大小、描边颜色、填充颜色？最后，怎么知道画布上其他形状的信息——这些信息在需要改变矩形和重绘画布时有用？

要解决这些问题，把Canvas变得具有交互性，必须记录绘制的每一个对象。此外，在用户单击Canvas中的某个地方时，还要检测被单击的是不是其中一个图形（这个过程叫*碰撞检测*）。如果能实现这两个任务，剩下的（修改某个图形或重绘画布）就简单了。

9.3.1 记录绘制的内容

为了修改和重绘画面，必须先知道要修改和重绘什么内容。就以图9-9所示的绘制圆圈的程序为例，但为了简单起见，其中包含的圆圈的大小、颜色各不相同。

为了记录每一个圆圈，需要知道它们的位置、半径以及填充色。与其声明一大堆变量来保存这些信息，不如把上述4个值都放在同一个小数据结构中。这个数据结构就是*自定义对象*。

什么，不知道怎么创建自定义对象？下面就是一种标准的做法。首先，创建一个函数，函数名就是用于创建这种对象的类型名。比如，要创建一种圆圈类型，可以把函数命名为`Circle()`：

```
function Circle() {
}
```

然后，需要让这个对象能保存数据。为此，要使用关键字`this`来创建属性。比如，要为将来的对象创建一个`radius`属性，以便记录它的大小，应该在`Circle()`函数中给`this.radius`赋一个起始值。

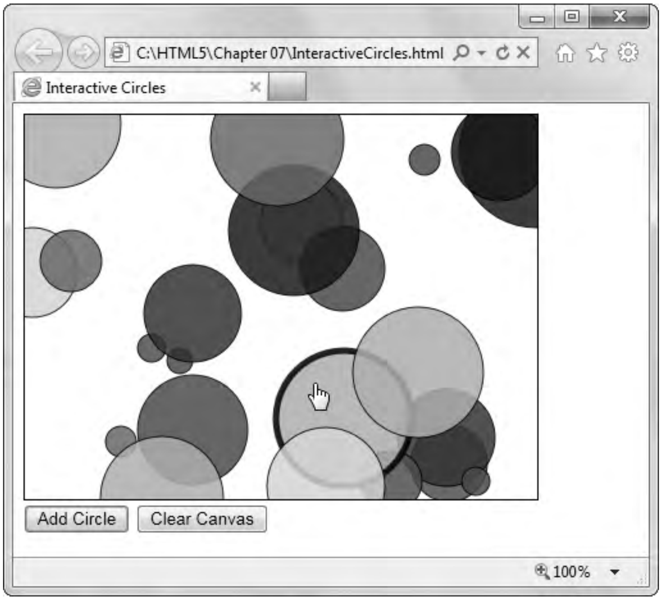


图9-9：这个绘制圆圈的程序是交互性的。单击可以选择一个圆（边框会变成另一种颜色），而且能把它拖动到新位置

下面这个函数用3个属性定义一个圆——*x*坐标、*y*坐标和半径：

```
function Circle() {
  this.x = 0;
  thix.y = 0;
  this.radius = 15;
}
```

Circle()函数里的数字都是默认值。用这个函数新建一个圆圈对象时，可以修改每个属性的值。

添加了所有所需的属性之后，就可以用这个函数创建对象了。这里就是使用Circle()函数来创建新的圆圈对象了。这里的关键是并非要调用该函数，而是要使用new关键字来创建它的一个副本，比如：

```
//创建一个新的Circle对象，并将其保存在变量myCircle中
var myCircle = new Circle();
```

有了这个圆圈对象，就可以像下面这样来访问它的属性：

```
//修改半径
myCircle.radius = 20;
```

不仅如此，要是你想让定义新圆圈对象的过程更灵活一些，还可以为Circle()函数传递参数。这样就可以在创建新圆圈对象的时候，一次性设置圆圈的所有属性。下面就是用于创建图9-9中圆圈对象的另一个Circle()函数：

```
function Circle(x, y, radius, color) {
  this.x = x;
  this.y = y;
```

```
  this.radius = radius;
  this.color = color;
  this.isSelected = false;
}
```

这里的isSelected属性值不是true就是false。在用户单击这个圆圈时，isSelected的值就会变成true，而此时绘图代码就知道应该为它绘制一个不同的边框了。

使用这个Circle()函数，可以利用如下代码来创建一个圆圈对象：

```
var myCircle = new Circle(0, 0, 20, "red");
```

当然，圆圈绘图程序最终是要支持用户画任意圆圈的。所以不可能只创建一个圆圈对象。为此，需要创建一个数组，用于保存所有圆圈。下面就是我们这个例子中所要用到的全局数组变量：

```
var circles = [];
```

剩下的代码也不难。在用户单击Add Circle按钮创建新的圆圈时，就会触发addRandomCircle()函数。addRandomCircle()函数会以随机大小、颜色和坐标值绘制一个圆圈：

```
function addRandomCircle() {
  //为圆圈计算一个随机大小和位置
  var radius = randomFromTo(10, 60);
  var x = randomFromTo(0, canvas.width);
  var y = randomFromTo(0, canvas.height);

  //为圆圈计算一个随机颜色
  var colors = ["green", "blue", "red", "yellow", "magenta",
   "orange", "brown", "purple", "pink"];
  var color = colors[randomFromTo(0, 8)];

  //创建一个新圆圈
  var circle = new Circle(x, y, radius, color);

  //把它保存在数组中
  circles.push(circle);

  //重新绘制画布
  drawCircles();
}
```

以上代码也利用了另一个自定义函数randomFromTo()，它在某个范围内生成随机数。

（要查看全部代码，请访问http://prosetech.com/html5。）

最后一步当然就是基于当前圆圈的集合实际地在画布上绘图了。创建新圆圈后，addRandomCircle()调用了另一个函数drawCircles()来执行绘图操作。drawCircles()函数会遍历圆圈数组，像下面这样：

```
for(var i=0; i<circles.length; i++) {
  var circle = circles[i];
  ...
}
```

以上代码使用可靠的for循环（详细介绍请参考附录B）。花括号中的代码块针对每个圆圈都

会运行一次。第一行代码先从数组中取得当前圆圈，将其赋给一个变量，以方便后面使用。

下面就是drawCircles()函数的完整代码，它的任务就是根据当前圆圈的集合来填充画布：

```
function drawCircles() {
  //清除画布，准备绘制
  context.clearRect(0, 0, canvas.width, canvas.height);
  context.globalAlpha = 0.85;
  context.strokeStyle = "black";

  //遍历所有圆圈
  for(var i=0; i<circles.length; i++) {
    var circle = circles[i];

    //绘制圆圈
    context.beginPath();
    context.arc(circle.x, circle.y, circle.radius, 0, Math.PI*2);
    context.fillStyle = circle.color;

    context.fill();
    context.stroke();
  }
}
```

注意 圆圈绘图程序每次刷新画布，都会先使用clearRect()方法清除画布上的所有内容。有些极其追求完美的程序员就担心这一步操作会造成画布闪烁，即画布上的圆圈一下全都消失，然后一下子又重新出现。不过，Canvas针对这个问题进行了优化。换句话说，它实际上会在绘图逻辑执行完毕后才清除或绘制**所有内容**，因此可以把最终结果流畅不间断地复制到画布上。

现在，圆圈仍然还没有交互性。不过，页面中用于记录绘制的每个圆圈的代码已经齐备了。尽管画布看上去仍然还是彩色像素块，但我们的代码知道画布所有圆圈的精确信息。而这就意味着可以随时操作这些圆圈。

下一节，我们就来看看如何在此基础上让用户选择圆圈。

9.3.2 基于坐标的碰撞检测

只要创建交互图形，几乎就一定要用到*碰撞检测*，也就是测试某个点是否“碰到”了某个图形。在绘制圆圈的程序中，我们需要检测用户单击的点是否碰到某个圆圈，或者只是点击了空白区域。

一些比较完善的动画开发框架（比如Flash或Silverligh）可以帮你做碰撞检测。虽然也有一些针对Canvas的JavaScript库（如KineticJS）能提供这种方便，但在本书编写时还没有哪个足够成熟，所以在此就不推荐了。所以，Canvas粉丝们最好的选择就是学会自己写碰撞测试代码（做完之后，可以考虑用9.4.2节附注“忙人（或懒人）的Canvas动画”中所提到的JavaScript库实践一把Canvas）。

为做碰撞测试，就要检测每一个形状，计算鼠标点击的那个点是否落在某个形状里。如果是，说明单击“碰到”了该形状。分析起来简单，而实现起来可就远没有那么容易了。

第一件事儿就是遍历所有形状。这个循环与前面drawCircles()函数所用的循环有一点不同：

```
for (var i=circles.length-1; i>=0; i--) {
  var circle = circles[i];
  ...
}
```

不同之处是这里的代码在反向遍历数组：从末尾开始（末尾的索引等于数组中包含的元素数减1），向开头迭代（第一个元素的索引为0）。这里的反向遍历是有意为之的，因为在大多数应用中（包括我们这个），都会按照数组中列出对象的顺序来绘制对象。结果，后来的对象可能就会叠加在先前对象上面。而在两个形状叠加起来后，那么单击的只能是上面的那个对象。

要确定单击点是否位于形状内，需要一些数学计算。对于圆圈而言，需要计算单击点与圆心的直线距离。如果这个距离小于等于圆圈半径，那么就可以确定单击点位于圆圈内。

在我们这个例子中，页面会处理Canvas的onClick事件，以检测被单击的圆圈。当用户单击画布时，就会触发canvasClick()函数。这个函数会取得单击点的坐标，然后检测该坐标是否位于某个圆圈内：

```
function canvasClick(e) {
  //取得画布上被单击的点
  var clickX = e.pageX - canvas.offsetLeft;
  var clickY = e.pageY - canvas.offsetTop;

  //查找被单击的圆圈
  for (var i=circles.length; i>0; i--) {
    //使用勾股定理计算这个点与圆心之间的距离
    var distanceFromCenter =
    Math.sqrt(Math.pow(circle.x - clickX, 2) + Math.pow(circle.y - clickY, 2))

    //这个点在圆圈中吗
    if (distanceFromCenter <= circle.radius) {
      //清除之前选择的圆圈
      if (previousSelectedCircle != null) {
        previousSelectedCircle.isSelected = false;
      }
      previousSelectedCircle = circle;

      //选择新圆圈
      circle.isSelected = true;

      //更新显示
      drawCircles();

      //停止搜索
      return;
    }
  }
}
```

注意 在9.5.3节创建迷宫游戏时，我们还会看到另一种碰撞检测：取得原始像素，比较它们的颜色。

这个例子的最后，要稍微调整一下drawCircles()函数中的代码。现在，应该为被选择圆圈加点标记，让它突出出来（如前所述，这里会加上一个粗边框）：

```
function drawCircles() {
  ...

  //循环所有圆圈
  for(var i=0; i<circles.length; i++) {
    var circle = circles[i];

    if (circle.isSelected) {
      context.lineWidth = 5;
    }
    else {
      context.lineWidth = 1;
    }
    ...
  }
}
```

这个例子当然还可以更加完善，功能更强大。例如，可以添加一个命令工具条，用于修改圆圈（修改颜色或把它从画布上删除）。或者，允许用户在画布上拖动圆圈。为此，只要侦听Canvas的onMouseMove事件，相应地修改圆圈的坐标，然后再调用drawCircles()函数重绘画布即可。（这其实就是8.2节讨论的简单画图应用所使用的技术，只不过现在是基于鼠标移动来画图，而非画线。）本书试验站点（http://prosetech.com/html5）中的InteractiveCircles_WithDrag.html页面，包含了一个演示这种技术的例子。

记住这个要点：只有记录绘制的所有内容，才能在将来灵活地修改并重绘它们。

9.4 给 Canvas 添加动画

绘制一幅完美的图画已经够复杂的了，所以就算是经验丰富的开发人员，让他实现每秒绘制几十个图形的程序，也难免不眉头紧锁。做动画的关键是绘制和重绘画布的速度要足够快，这样才能让人感觉移动和变化自然流畅。

动画对某些应用来说可以是最基本的，比如实时游戏、物理模拟器。不过，比较简单的动画在包含Canvas的页面中同样大有用武之地。可以通过动画来突出用户交互（例如，在鼠标悬停时，给图形加上光晕、让图形跳动或闪烁），也可以利用动画效果来吸引人注意改变的内容（例如，淡入新场景，或创建“长”到恰当位置的图形、图表）。如此说来，动画确实是为网页增光增彩的强大手段，能给人活生生的感觉，而且还能帮我们从一大堆竞争者中脱颖而出。

9.4.1　基本的动画

在HTML5中利用Canvas实现动画非常容易。首先，要设置一个定时器，反复调用绘图函数（一般每秒30～40次）。每次调用，都会重绘整个画布。完成后的效果就像动画一样，每一帧间的过渡会平滑而流畅。

JavaScript为控制重复绘制提供了两种手段。

- **使用setTimeout()函数**。这个函数告诉浏览器等待多长时间（毫秒），然后再运行一段代码（即绘制画布的代码）。运行代码后，可以再调用setTimeout()让浏览器准备下一次运行。如此往复，直至动画结束。
- **使用setInterval()函数**。这个函数告诉浏览器每隔一定时间（如20毫秒）就运行某一段代码。它与setTimeout()的效果类似，但只需调用setInterval()一次。要阻止浏览器继续运行代码，可以调用clearInterval()。

假如运行绘图代码的速度非常快，使用这两个函数都可以，结果都一样。可假如绘图代码没那么快，setInterval()则能保证精确地按时重绘，但又可能因此牺牲性能。（最差的情况下，如果绘图代码执行时间比设定的时间还要长，浏览器将很难跟上，随着绘图代码连续执行，页面会出现短暂地停顿。）考虑到这个原因，本章的例子都使用setTimeout()函数。

调用setTimeout()时，要提供两个参数：要运行的函数名和运行该函数之前等待的时间。这里的时间要使用毫秒（千分之一秒），因此20毫秒（典型的动画延迟时间）就是0.02秒。来看下面这个例子：

```
var canvas;
var context;

window.onload = function() {
  canvas = document.getElementById("canvas");
  context = canvas.getContext("2d");

  //每0.02秒绘制一次画布
  setTimeout(drawFrame, 20);
};
```

任何动画的关键都在于调用setTimeout()。例如，要想编写一个方形从上到下坠落的动画，就需要像下面这样用两个全局变量跟踪方形的位置：

```
//设置方形的初始位置
var squarePosition_y = 0;
var squarePosition_x = 10;
```

接下来，只要在每次调用drawFrame()函数时改变方形的位置，然后在新位置重绘方形即可：

```
function drawFrame() {
  //清除画布
  context.clearRect(0, 0, canvas.width, canvas.height);

  //调用beginPath()，确保不会接着上次绘制的图形绘制
```

```
    context.beginPath();

    //在当前位置绘制10像素×10像素的方形
    context.rect(squarePosition_x, squarePosition_y, 10, 10);
    context.lineStyle = "black";
    context.lineWidth = 1;
    context.stroke();

    //向下移动1像素（下一帧将在此位置绘制）
    squarePosition_y += 1;

    //20毫秒后绘制下一帧
    setTimeout(drawFrame, 20);
}
```

运行这个例子，就会看到一个方形从画布上方不断下落，最后消失在画布下方。

如果动画更复杂，计算过程也会相应复杂。比如，要模拟重力加速度，或者模拟方形撞击底边后反弹。但“设置计时器、调用绘制函数和重绘整个画布”这个基本过程都是完全相同的。

9.4.2 多物体动画

好了，既然都介绍了动画和交互绘制画布的基本知识，下面我们就更进一步，把这些知识综合起来运用到一个例子中。图9-10展示了一个测试页面，其中有多个下落和弹跳的球。这个例子使用了上一节用到的`setTimeout()`方法，而此次绘制代码必须支持无数个下落的小球。

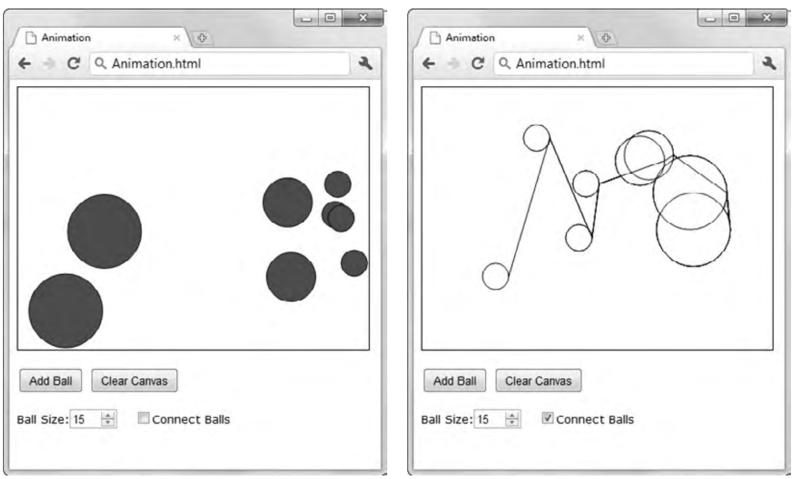


图9-10：在这个测试页面中，可以添加任意多个球，可以选择球的大小（默认半径为15像素），还可以打开连线功能（右图）。添加每个球之后，这个球就会独立运动，加速向下坠落，直至碰到画布底边弹回来

提示 静态图片当然反映不出动画效果了。要尝试一下图9-10所示的动画，可以访问http://prosetech.com/html5。

动画的性能问题

由于绘制速度很快，因此与基本的绘图操作相比，动画对画布的要求要高得多。但出人意料的是，画布并没有反应迟钝。这是因为现代浏览器都使用了**硬件加速**等性能增强技术，把图形处理工作转移给了显卡，从而节省了CPU。即使JavaScript不是现有最快的语言，但仍然可以利用它来创造出复杂、高速的动画，甚至是实时电子游戏——只要有脚本和画布即可。

然而，对于移动设备（比如iPhone或Android手机）来说，由于能力不足，性能就是一个问题了。测试表明，在桌面浏览器中运行速度达每秒60帧的动画，在智能手机中最高才能达到每秒10帧。因此，要是想为手机用户开发应用，一定要尽早测试并准备牺牲一些夺人眼目的动画效果，从而确保应用运行流畅。

要管理图9-10中这些球，需要用到9.3.1节讨论的自定义对象。只不过现在需要记录很多球对象，而每个球对象不仅要有位置（属性x和y），还要有速度（属性dx和dy）：

```
//下面就是用于表示球的所有细节的Ball函数
function Ball(x, y, dx, dy, radius) {
  this.x = x;
  this.y = y;
  this.dx = dx;
  this.dy = dy;
  this.radius = radius;
  this.color = "red";
}

//这个数组用于保存画布上出现的所有球
var balls = [];
```

注意 用数学书里的说法，dx就是x改变的速度，而dy就是y改变的速度。因此，随着球的下落，每一帧x都会增加dx，而y都会增加dy。

用户单击Add Ball按钮后，几行代码会执行：创建一个新的ball对象并将其保存在balls数组中：

```
function addBall() {
  //取得用户设定的大小
  var radius = parseFloat(document.getElementById("ballSize").value);

  //创建新的ball对象
  var ball = new Ball(50,50,1,1,radius);

  //将其保存在balls数组中
  balls.push(ball);
}
```

Clear Canvas按钮的任务恰恰相反——清空balls数组：

```
function clearBalls() {
  //删除所有球对象
  balls = [];
}
```

可是，addBall()和clearBalls()函数实际上都不会绘制图形；它们都没有调用绘制函数。实际上，调用drawFrame()函数的代码会在页面加载后计时执行，每隔20毫秒就重绘一次画布：

```
var canvas;
var context;

window.onload = function() {
  canvas = document.getElementById("canvas");
  context = canvas.getContext("2d");

  //每20毫秒重绘一次
  setTimeout(drawFrame, 20);
};
```

drawFrame()函数是这个例子的关键所在，它不仅负责在画布上绘制所有球，而且还要计算每个球的当前位置和速度。为此，drawFrame()函数使用了一些计算方法模拟真实的运动。比如，坠落时加速，而反弹时减速。下面就来看看它的完整代码：

```
function drawFrame()   //清除画布
  context.clearRect(0, 0, canvas.width, canvas.height);
  context.beginPath();

  //循环所有球
  for(var i=0; i<balls.length; i++) {
    //把每个球移动到新位置
    var ball = balls[i];
    ball.x += ball.dx;
    ball.y += ball.dy;

    //添加重力作用的效果，让球加速下落
    if ((ball.y) < canvas.height) ball.dy += 0.22;

    //添加摩擦力作用的效果，减慢左右移动速度
    ball.dx = ball.dx * 0.998;

    //如果球碰到某一边，就反弹回来
    if ((ball.x + ball.radius > canvas.width) || (ball.x - ball.radius < 0)) {
      ball.dx = -ball.dx;
    }

    //如果球碰到底部，反弹回来，但慢慢地减速
    if ((ball.y + ball.radius > canvas.height) || (ball.y - ball.radius < 0))
    {
      ball.dy = -ball.dy*0.96;
    }
```

```
    //检测用户是否选择了连线功能
    if (!document.getElementById("connectedBalls").checked) {
      context.beginPath();
      context.fillStyle = ball.fillColor;
    }
    else {
      context.fillStyle = "white";
    }

    //绘制球
    context.arc(ball.x, ball.y, ball.radius, 0, Math.PI*2);
    context.lineWidth = 1;
    context.fill();
    context.stroke();
  }

  //20毫秒后绘制下一帧
  setTimeout(drawFrame, 20);
}
```

提示　如果有读者不太理解if语句的作用，不太清楚!和||等操作符的意思，可以参考附录B。

一下子看到这么多代码，是不是有点害怕？别紧张，整体流程并没有变。以上代码执行了下列任务：

(1) 清除画布；

(2) 循环球的数组；

(3) 调整每个球的位置和速度；

(4) 绘制每个球；

(5) 调用`setTimeout()`以便每隔20毫秒就执行一次`drawFrame()`函数。

其中第3步相对最复杂，因为球的属性是在这一步改变的。根据要实现的效果，这里的代码可能比现在还要复杂很多倍。渐进地、自然地运行非常难模仿，因此往往需要很多数学计算。

最后，既然每个球的状态都有记录，那么接下来为画布添加交互功能也就不难了。实际上，这里仍然可以使用9.3.2节中的碰撞检测代码，只不过此时在单击球的时候，需要以其他方式给出响应。比如，可以让被单击的球突然加速，让它向某个方向弹开。（实现这种响应的示例页面从http://prosetech.com/html5下载。）

要想知道这个例子做到极致是什么样的，可以看看这个谷歌弹跳球：http://tinyurl.com/6byvnk5。在没有鼠标介入的情况下，这些球就像有磁性一样拼成“Google”字样。在鼠标移动到其中后，小球像是受到了排斥，向画布的四周扩散，然后不规则地反弹回来。如果看了这个例子还不满足，再推荐两个给你：一个是迟钝的水滴（http://www. blobsallad.se），另一个是有点老套的星空效果（http://tinyurl.com/crn3ed）。

9

忙人（或懒人）的Canvas动画

难道所有计算都要我自己费脑子吗？

画布最大的缺点就是一切都需要你自己费脑子。比如，要想让一张图片从画布一边飞到另一边，你得自己计算图片在每一帧中的位置，然后再在该位置上绘制图片。而如果同时要以不同方式为不同的东西添加动画，那么代码就会变得很杂乱。相对而言，使用Flash或Silverlight的程序员的日子就好过一些了。这两种技术都内置了动画机制，开发人员只要给出类似“用45秒时间把这个图形从这儿移动到那儿”之类的指令即可。甚至可以说：“把这个图形从窗口顶部移动到底部，要加速下落，反弹时应该温和一些。”

为填补这个空缺，一些进取的JavaScript开发者们开始创建基于Canvas的高级绘图和动画系统。使用这些JavaScript库，可以选择想要的效果，而不必费尽心机地算计数字。有什么坑呢？目前至少有6个高质量的Canvas动画框架，每个都有其独特的模式和一些奇特之处。现在还说不好这些发展中的工具哪个在将来会支持得最好，最受欢迎。目前最值得推荐的的框架有：Fabric.js（http://fabricjs.com）、Paper.js（http://paperjs.org）、EaselJS（www.createjs.com）和KineticJS（http://kineticjs.com）。可以在http://tinyurl.com/canvas-libraries查看开发者们对这些库的最新评论。

9.5 实例：迷宫游戏

到目前为止，我们已经学习了针对画布编程的基本技术，学习了画布的交互功能和动画效果。运用这些基本的技术，不仅仅能绘图，而且可以实现完整的应用，比如游戏或Flash风格的迷你应用等。

图9-11展示了一个更有挑战性的例子，这个例子利用了迄今所学，包括两个概念。实际上，这是一个简单的游戏，让用户引导一个可爱的小笑脸图标走出迷宫。用户按下方向键时，笑脸图标会沿相应方向移动（动画），遇到墙时（碰撞检测）就会停下来。

当然，一分耕耘一分收获。要想利用画布实现这种高级应用，就得多编写很多代码。接下来的几节会详细介绍这个例子的开发过程，不过在此之前，建议你多学点JavaScript备用。

注意 如果你使用IE9，可以在本地计算机中运行这个例子。如果你使用其他浏览器，那么只有把页面（和其中用到的图像）上传到Web服务器才行。为节省时间，也可以到http://prosetech.com/html5直接查看这个例子。

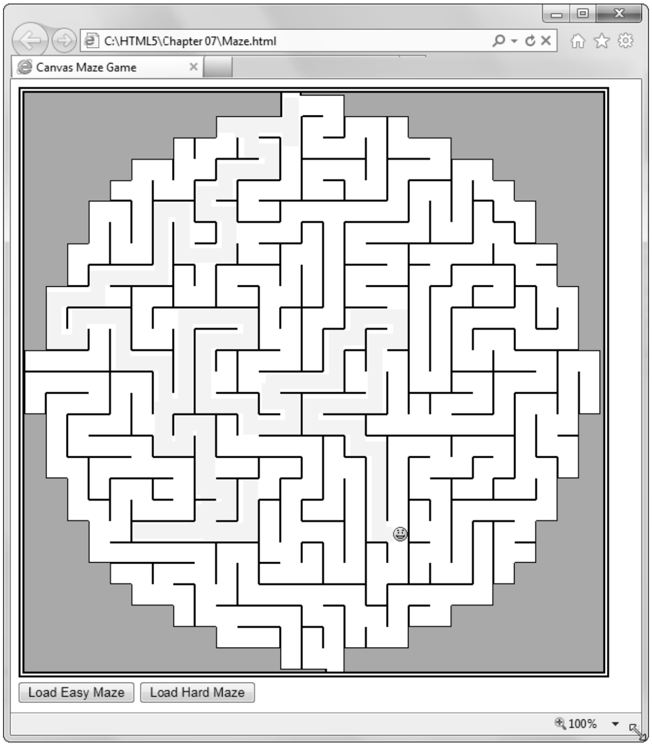


图9-11：引导笑脸走出迷宫。对于用户来说，这是个好玩的游戏。对于开发者来说，可以借以熟悉HTML5的Canvas和JavaScript编程技巧

9.5.1 布置迷宫

在一切发生之前，需要在页面中设置画布。当然可以手工绘制迷宫的线条或矩形，但这样就需要编写很多代码。手工编写这些代码极其烦琐。你得在大脑中想象一个迷宫，然后再用独立的绘图操作绘制每一堵墙。要是你真打算这样做，应该使用能够自动创建绘图代码的工具。例如，可以在Adobe Illustrator里绘图，然后使用插件导出画布代码（参见8.1.4节）。

另一种思路是选择一幅迷宫图片，把整幅图绘制到画布上。这个办法就简单多了，因为在网上可以找到很多能生成迷宫的免费页面。找到某个页面后，设置一些参数（如迷宫大小、形状、颜色、密度和复杂性），页面就能创建一个可下载的图片。（想试试？用谷歌搜索maze generator。）

我们这个例子使用迷宫图片。当页面加载时，它会取得一张图片（名为maze.png），然后把它绘制到画布上。以下就是实现上述过程的代码：

```
//定义全局变量，保存画布及绘图上下文
var canvas;
var context;

window.onload = function() {
  //设置画布
```

9

```
  canvas = document.getElementById("canvas");
  context = canvas.getContext("2d");

  //绘制迷宫背景
  drawMaze("maze.png", 268, 5);

  //当用户按下键盘上的键时，运行processKey()函数
  window.onkeydown = processKey;
};
```

以上代码并没有自己绘制迷宫背景，而是把这个任务交给了另一个名为drawMaze()的函数。

由于绘制迷宫的是一个独立的函数，因此就可以不局限于只绘制一个迷宫。只要在调用drawMaze()时传入迷宫图片的文件名以及笑脸的起始位置，就可以加载任何迷宫图片。下面就是绘制迷宫的drawMaze()函数：

```
//记录笑脸图标的当前位置
var x = 0;
var y = 0;

function drawMaze(mazeFile, startingX, startingY) {
  //加载迷宫图片
  imgMaze = new Image();
  imgMaze.onload = function() {
    //调整画布大小以适应迷宫图片
    canvas.width = imgMaze.width;
    canvas.height = imgMaze.height;

    //绘制迷宫
    var imgFace = document.getElementById("face");
    context.drawImage(imgMaze, 0,0);

    //绘制笑脸
    x = startingX;
    y = startingY;

    context.drawImage(imgFace, x, y);
    context.stroke();

    //10毫秒后绘制下一帧
    setTimeout(drawFrame, 10);
  };
  imgMaze.src = mazeFile;
}
```

以上代码使用了9.1.1节介绍的两步绘制图像的方法。首先，定义一个处理图片onLoad事件并在图片加载完毕后绘制迷宫的函数。其次，它设置了图片对象的src属性，这样就会加载图片并在加载完毕后触发事件处理函数。与从隐藏的<img>元素中取出图片相比，这个两步方法稍微复杂那么一点，但为了让函数足够灵活，可以加载任意迷宫图片，就必须采取这种方法。

加载完迷宫图片后，代码会根据图片大小调整画布的大小，把笑脸图标放到正确的位置上，然后绘制笑脸图标。最后，调用setTimeout()开始绘制动画帧。

注意 我们的试验网站（http://prosetech.com/html5）中的这个例子还要复杂一些，主要是支持用户在任何时候加载新的迷宫，即使笑脸图标在当前迷宫中行进期间也可以。为此，那个例子在drawMaze()函数中添加了一些代码，用于停止笑脸图标（如果正在行进的话），终止动画进程，然后重新加载背景，重新开始。

9.5.2 让笑脸动起来

在用户按下键盘上的方向键时，笑脸开始移动。比如，按向下键，笑脸就会一直向下移动，不是碰到障碍或用户又按了其他方向键，就不会停下来。

为此，我们在代码中需要使用两个全局变量记录笑脸的速度。换句话说，就是记录笑脸在*x*和*y*轴方向上每一帧要移动多少像素。这两个变量就是dx和dy，与上一节弹跳球例子中的一样。区别在于，这个例子不会用到数组，因为只有一个笑脸图标：

```
var dx = 0;
var dy = 0;
```

用户按下键盘上的键时，画布就会调用processKey()函数。然后，该函数检查用户按下的是不是方向键，然后据以调整笑脸的速度。为了检测方向键，要用已知的值与用户按下键的键码进行比较。比如，38是向上键的键码。processKey()函数会忽略除方向键之外的按键：

```
function processKey(e) {
  //如果笑脸在移动，停止
  dx = 0;
  dy = 0;

  //按下了向上键，向上移动
  if (e.keyCode == 38) {
    dy = -1;
  }

  //按下了向下键，向下移动
  if (e.keyCode == 40) {
    dy = 1;
  }

  //按下了向左键，向左移动
  if (e.keyCode == 37) {
    dx = -1;
  }

  //按下了向右键，向右移动
  if (e.keyCode == 39) {
    dx = 1;
  }
}
```

从代码中可见，processKey()函数并不改变笑脸的当前位置，也没有绘制笑脸。在调用了drawFrame()函数之后，每隔10毫秒就会执行一次这种检测任务。

接下来看drawFrame()函数，这个函数的代码很好理解，只是会涉及很多细节。这个函数执行几个任务，首先是检测笑脸是否正在哪个方向上移动。如果不是，则什么也不必做：

```
function drawFrame() {
  if (dx != 0 || dy != 0) {
```

如果笑脸在移动，drawFrame()会在当前笑脸的位置绘制一块黄色背景（用于创造“痕迹”感），然后把笑脸移动到下一个位置：

```
    context.beginPath();
    context.fillStyle = "rgb(254,244,207)";
    context.rect(x, y, 15, 15);
    context.fill()

    //增大位置值
    x += dx;
    y += dy;
```

接下来，调用checkForCollision()函数，检查新位置是否与障碍物冲突。（下一节将介绍这个碰撞检测函数的代码。）如果新位置无效，说明笑脸碰到了墙，代码必须将其放回上一位置并停止移动它：

```
    if (checkForCollision()) {
      x -= dx;
      y -= dy;
      dx = 0;
      dy = 0;
    }
```

到这里就可以绘制笑脸了，以下就是代码：

```
    var imgFace = document.getElementById("face");
    context.drawImage(imgFace, x, y);
```

然后，检测笑脸是否到已经达迷宫底部（完成了游戏）。如果是，则显示一个消息框：

```
    if (y > (canvas.height - 17)) {
      alert("You win!");
      return;
    }
  }
```

如果没有，则通过setTimeout()设置在10毫秒之后再次调用drawFrame()方法：

```
  //10毫秒后绘制下一帧
  setTimeout(drawFrame, 10);
}
```

除了checkForCollision()函数，这个例子的代码就都介绍完了。下面我们就来介绍用于碰撞检测的创新逻辑。

9.5.3 基于像素颜色的碰撞检测

本章前面曾讨论过基于数学计算来实现碰撞检测。除此之外，还有另一种手段。那就是不检测已经绘制了哪些对象，而是取得像素块，检测它们的颜色。很多情况下，这种手段更简单，因为它不涉及全部对象，也不必编写图形记录代码。然而，这种手段只适合能明确判断颜色的场合。

> **注意** 对于迷宫游戏来说，基于像素颜色进行碰撞检测是最理想的手段。利用像素颜色，可以确定笑脸什么时候碰到黑色的墙。如果不使用这种技术，那么就必须将迷宫的信息保存在内存中，然后再确定笑脸的当前坐标是否与迷宫中的某面墙重叠。

能够基于像素颜色进行碰撞检测的关键，就在于画布支持对个别像素（也就是组成每张图片的小点）的操作。绘图上下文为操作像素提供了三个方法：getImageData()、putImageData()和createImageData()。其中，getImageData()用于从矩形区域中取得一个像素块，然后再检测这些像素（我们的迷宫游戏中使用了这个方法）。可以修改像素使用putImageData()并将它们回写到画布。最后，createImageData()用于在内存中创建新的、空的像素块，以便你根据自己的想法自定义其中的像素，然后使用putImageData()把它们回写到画布上。

为了深入地理解这几个操作像素的方法，来看下面这行代码。这行代码首先从当前画布上取得一个100像素×50像素大的像素块，使用的是getImageData()方法：

```
//取得像素的起点为(0,0)，向右拓展100像素，向下拓展50像素
var imageData = context.getImageData(0, 0, 100, 50);
```

然后，再通过data属性取得一个包含图像数据的数值数组：

```
var pixels = imageData.data;
```

可能有读者会猜测每个像素都用数组中的一个数值表示。要是那么简单就好了！实际上，每个像素是用4个数值来表示的，前三个分别表示红、绿、蓝，第四个表示不透明度（alpha值）。因此，要检测每个像素，必须四个一组四个一组地遍历这个数组，如下所示：

```
//遍历每个像素，反转其颜色
for (var i = 0, n = pixels.length; i < n; i += 4) {

  //取得每个像素的数据
  var red = pixels[i];
  var green = pixels[i+1];
  var blue = pixels[i+2];
  var alpha = pixels[i+3];

  //反转颜色
  pixels[i] = 255 - red;
  pixels[i+1] = 255 - green;
  pixels[i+2] = 255 - blue;
}
```

每个数值的范围是0~255。上面的代码使用了最简单的图像操作技术——反转颜色。如果反

转的是一张照片，那么结果就像看到其负片一样。

为了看到反转颜色后的效果，可以把修改后的像素回写到画布中原来的位置上（当然，绘制到任何地方都易如反掌）：

```
context.putImageData(imageData, 0, 0);
```

能够操作每个像素，当然为我们控制画布提供了很多可能性。但是，操作像素也有缺点，主要是操作速度慢，而且一般画布中包含的像素数目都十分巨大。如果取得一大块图片数据，可能就需要遍历几万个像素。如果你觉得画直线或画曲线都很烦人，那么手工处理一个个像素只会更令人生厌。

但是，操作像素能够解决其他手段解决不了的问题。比如，通过像素操作可以方便地绘制出分形图像，或者实现Photoshop风格的图片滤镜。而在我们的迷宫游戏中，通过像素操作可以用简单的代码来确定笑脸图标的走向，判断它是否碰到了墙。以下就是处理这个任务的checkForCollision()函数的代码：

```
function checkForCollision() {
  //取得笑脸所在的像素块，再稍微扩展一点
  var imgData = context.getImageData(x-1, y-1, 15+2, 15+2);
  var pixels = imgData.data;

  //检测其中的像素
  for (var i = 0; n = pixels.length, i < n; i += 4) {
    var red = pixels[i];
    var green = pixels[i+1];
    var blue = pixels[i+2];
    var alpha = pixels[i+3];

    //检测黑色的墙（如果检测到了，就说明撞墙了）
    if (red == 0 && green == 0 && blue == 0) {
      return true;
    }

    //检测灰色的边（如果检测到了，就说明撞墙了）
    if (red == 169 && green == 169 && blue == 169) {
      return true;
    }
  }
  //没有碰到墙
  return false;
}
```

这样，我们迷宫游戏的代码就全部介绍完了。这个例子是本书到目前为止代码最长的一个例子。要想完全理解它们，恐怕得多看几遍（或者好好掌握一下JavaScript），而一旦你看懂看明白了它们，就可以在自己的画布应用中用上类似的技术了。

如果你渴求更多的Canvas教程，可能需要一门专门介绍这个主题的书，比如*HTML5 Canvas*（Steve Fulton和Jeff Fulton著）或者*Core HTML5 Canvas*（David Geary著）。每本书都可以带你深入Canvas绘图的底层细节。

有视觉冲击力的Canvas应用示例

利用Canvas，可以开发出不计其数的应用形式。如果你想知道通过HTML5画布到底能实现多么神奇的效果，可以在网上找。下面也给出几个网站，在其中可以发现一些令人目眩的画布魔法。

- **Canvas Demos**。这个Canvas示例网站中的内容足以让人流连忘返。本书写作时的例子有游戏*Mutant Zombie Masters*《突变僵尸大师》和股份图工具TickerPlot。不说了，赶紧看吧：http://www.canvasdemos.com。
- **维基知识地图**。这个令人难忘的画布应用，以图形方式形象地展示了维基百科中的文章。不同主题由网状的细线相连。选择其中一个主题，就可以进入放大的知识地图中，然后流畅的动画会把最新的文章展示出来。这个网站的地址是http://en.inforapid.org。
- **3D Walker**。这个例子展示的是一个简单的3D围墙和通道环境（让人仿佛置身二战时期的德军总部，当然是指那款1992年发布的开第一人称射击游戏先河的大作）。想试一试吗？请访问http://www.benjoffe.com/code/demos/canvascape。
- **国际象棋**。这是一个HTML5国际象棋模拟程序，你可以跟计算机对弈（上方选手），可以使用三维视角，取决于你的设置。要挑战自己？来吧：http://htmlchess.sourceforge.net/demo/example.html。

Part 3

第三部分

构建 Web 应用

本部分内容

第 10 章

数据存储

Web应用中的数据可以保存到两个地方，一个是Web服务器，一个是Web客户端（用户的计算机）。这两个地方各自都有适合保存的数据。

Web服务器适合保存敏感信息，以及那些你不希望被别人篡改的数据。比如，在网上书店里购书时，所有交易信息都会保存到Web服务器上，比如图书类别、历史消费记录以及其他任何相关信息。你自己计算机中保存的数据只有那么一丁点儿，书店网站要用该数据判断你是谁，这样才能知道哪个购物车是你的。即使有了HTML5，这种套路也不能改变——服务器安全、可靠、高效。

然而，服务器端存储也并非适合所有网站。有时候，把一些不太重要的信息放在用户计算机上相对会更方便。比如，把*用户偏好*（影响网页显示方式的设置）和*应用状态*（相当于Web应用某个瞬间的快照，保存后可以方便用户将来返回该状态）放在用户本地就比较合乎情理。如果从服务端获取或计算数据比较耗时，可以通过将其保存在用户本地来提供性能。

在HTML5之前，本地存储的唯一方案就是使用cookie。而最初发明cookie的目的，是为了在浏览器和服务器之间传送身份信息。利用cookie保存少量数据绝对方便，可是操作它们的JavaScript语法多少有点不够人性化。但cookie也有不好的一面，那就是必须处理过期数据，而且要跟着每一次请求来来回回地发送和接收这些没有用的数据。

HTML5新增了更好的本地存储功能，让我们在访客的计算机上保存数据更加方便。这些数据可以无限期地保存在用户计算机上，不会发送到服务器（除非你自己发送），有充裕的空间存储它们，而且能够通过几个简单的JavaScript对象对它们进行操作。这个叫作*Web存储*（Web Storage）的新功能特别适合开发离线应用（离线应用将在第11章讨论）。离线应用的数据可以“自给自足”，无论用户能否上网，都可以在本地保存用户信息。

本章将带领大家探索Web存储功能的各个方面。另外，还会介绍两个更新的标准：File API，支持该标准的浏览器能够从计算机硬盘的其他文件中读取数据；IndexDB，支持该标准的浏览器内含一个完整的、微型的数据库引擎。

10.1 Web 存储简介

HTML5的这个Web存储功能，其实就是让网页在用户计算机上保存一些信息。这些信息可以是临时的（浏览器一关，就自动删除），也可以是长期的（多少天之后再打开网站，仍然可以访问它们）。

注意 **Web存储**这个名字误人不浅啊，本来网页保存的信息根本就不在Web（网）上，而是实实在在地保存在用户的计算机上，从来不会离开。

Web存储又分两种，分别对应两个JavaScript对象。

- **本地存储**，对应`localStorage`对象，用于长期保存网站的数据，并且站内任何页面都可访问该数据。也就是说，如果有一个网页利用本地存储保存了数据，那么访客在一天后、一星期后，甚至一年之后再上线，该数据仍然还会在那儿。当然，多数浏览器都会提供一种机制，让用户可以清除本地存储空间中的数据。有些浏览器只提供一个命令，要么一点不能删，要删就全都删除，就跟过去清除cookie的方法一样。（事实上，某些浏览器的实现是将本地存储和cookie放在一起的，因此要清除本地存储数据必须清除cookie。）另一些浏览器会让用户按照站点检查数据，然后有选择地清除本地存储的数据。
- **会话存储**，对应`sessionStorage`对象，用于临时保存针对一个窗口（或标签页）的数据。在访客关闭窗口或标签页之前，这些数据是存在的，而关闭之后就会被浏览器删除。不过，只要用户不关闭窗口或标签页，就算他从你的网站跑到人家的网站然后又回来，这些数据还会在。

提示 从页面代码的角度说，本地存储和会话存储的操作完全相同。它们的区别仅在于数据的寿命。本地存储主要用于保存访客将来还能看到的数据，而会话存储则用于保存那些需要从一个页面传递给下一个页面的数据。（当然,使用会话存储也可以保存只在一个页面中使用的数据，但这个任务就算普通的JavaScript变量也绝对可以胜任，又何必多此一举呢。）

无论本地存储还是会话存储，都是与网站所在的域联系在一起的。换句话说，如果利用本地存储保存数据的页面是www.GoatsCanFloat.org/game/zapper.html，那么另一个页面www.GoatsCanFloat.org/contact.html也可以访问该数据，因为这两个页面在同一个域中（www.GoatsCanFloat.org）。如果是另一个不同网站的页面，就不能访问该数据了。

同样，由于数据保存在用户的计算机上（或移动设备上），这些数据也是跟计算机绑定的；网页不能访问保存在其他用户计算机上的数据。类似地，如果你用不同的用户名登录自己的计算机，或者使用不同的浏览器，那么存取的也将是不同的本地存储数据。

注意 尽管HTML5没有硬性规定存储空间的上限，但大多数浏览器都把本地存储限制为5 MB以下。这已经足够保存很多数据了，但如果你想利用本地存储来缓存大图片或视频文件，恐怕这个限制会让你捉襟见肘（毕竟，这也不是设计本地存储的目的）。假如你需要更大的空间，可以考虑尚未定案的IndexedDB数据库标准（参见10.4节的附注栏）——用户同意的前提下可以达到几百兆。

10.1.1　存储数据

要把一段信息保存到本地存储或会话存储中，首先要为该信息想一个名字。这个名字叫作键，将来要通过它来取回数据。

要保存数据，需使用localStorage.setItem()方法：

```
localStorage.setItem(keyName, data);
```

举个例子，假设你想保存用户名，那么这个键就可以叫作user_name：

```
localStorage.setItem("user_name", "Marky Mark");
```

当然，像这样保存硬编码的数据没有多大意思。更多情况下，可以保存动态数据，比如当前日期、数学计算的结果，或者用户在文本框中输入的某些文本，等等。下面就是一个保存动态数据的例子：

```
//取得文本框
var nameInput = document.getElementById("userName");

//保存文本框中的文本
localStorage.setItem("user_name", nameInput.value);
```

读取本地存储中的数据跟保存数据一样简单，只需使用localStorage.getItem()方法。例如，下面这行代码会读出前面保存的用户名，然后通过警告框显示出来：

```
alert("You stored: " + localStorage.getItem("user_name"));
```

无论这个名字是5秒钟前保存的，还是5个月前保存的，这行代码都管用。

当然，有可能这个键下面尚未保存任何数据。要检测某个键的值是否为空，可以直接测试它是否等于null。请看下面的例子：

```
if (localStorage.getItem("user_name") == null) {
  alert ("You haven't entered a name yet.");
}
else {
  // 把用户名放到文本框中
  document.getElementById("userName").value = localStorage.getItem("user_name");
}
```

会话存储也一样简单。唯一的区别是要使用sessionStorage对象，而不是localStorage对象：

```
//取得当前日期
var today = new Date();

//以文本形式保存格式为HH:mm的时间
var time = today.getHours() + ":" + today.getMinutes();
sessionStorage.setItem("lastUpdateTime", time);
```

图10-1展示了一个页面，利用了刚才介绍的这些概念。

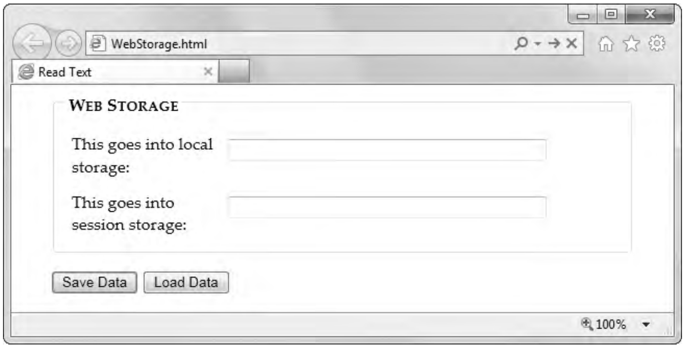


图10-1：页面中有两个文本框，一个针对会话存储，另一个针对本地存储。单击Save Data，页面会保存方框中的值。再单击Load Data，又会把刚才保存的数据读回来。要测试这个页面（同时验证会话存储中的数据会在关闭窗口后消失，而本地存储中的数据则会长期存在），可以运行http://prosetech.com/html5中的这个页面

注意 Web存储还支持两种读写数据的语法。除了前面介绍的getItem()和setItem()，还可以使用属性名或索引方式。用属性名时，读取名为user_name的数据段的方式为localStorage.user_name；用索引方式时，读取这个数据段的方式为localStorage["user_name"]。可以根据个人喜好选择读写语法，但大多数Web专家认为getItem()和setItem()最好，因为歧义最少。

没有Web服务器则不能使用Web存储

在测试 Web 存储时，可能会遇到一个意想不到的问题。在很多浏览器中，只有从 Web 服务器上打开的页面才能读写 Web 存储。无论这个 Web 服务器是远程的还是本地的——关键就是不能从本地硬盘打开页面。

这个问题的根源在于浏览器要限制 Web 存储的空间大小。如前所述，每个网站的存储上限是 5 MB。为了达到这一目的，就必须把每个使用本地存储的页面与一个网站（域）关联起来。

那如果我就是从本地硬盘打开一个使用 Web 存储的页面，结果会怎么样？结果要视情况而定。在 Internet Explorer 中，浏览器好像会完全不支持 Web 存储功能一样。因为，localStorage 和 sessionStorage 对象不见了，访问它们的 JavaScript 代码会报错。在 Firefox 中，localStorage 和 sessionStorage 对象还在，似乎还支持 Web 存储（即使 Modernizr 也会认为支持），但任何读写操作都会静默地失败。而在 Chrome 中，结果又不一样——大多数 Web 存储的功能一如既往，只有少数功能（如 onStorage 事件）无效。在使用 File API 的时候（10.3 节会介绍），也会遇到类似的问题。因此，在测试之前最好先把页面放到你自己的 Web 服务器中，从而避免意外发生。要不，直接运行 http://prosetech.com/html5 中的页面也可以。

10.1.2 实战：保存游戏中的最后位置

此时此刻，或许你会想：Web存储不过如此啊，也就是能用方括号语法记录人名而已。你说的没错。但Web存储还有更实际的用途，而且照样不必多费力气。

以我们第9章讲画布时介绍的迷宫游戏为例（见9.5节）。要想一次就走出迷宫并不容易，所以有必要在用户关闭窗口或打开新页面时记录当前位置。这样，用户再回来迷宫页面后，就可以把笑脸图标放在上一次的位置上。

要实现这个功能，有几种方案可以选择。可以每移动一次就保存一次新位置。本地存储的速度非常快，因此这样做没有问题。或者，可以响应页面的onBeforeUnload事件，询问游戏玩家是否要保存当前位置（如图10-2所示）。

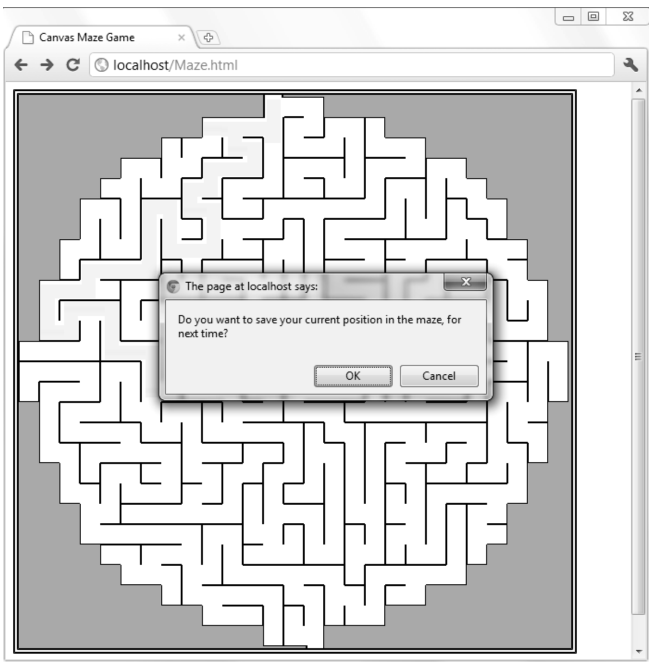


图10-2：在玩家因为要打开新页面或关闭窗口而离开当前页面时，页面建议保存当前位置

以下就是实现页面建议保存位置的代码：

```
window.onbeforeunload = function(e) {
  //检测localStorage对象是否存在
  //（如果浏览器不支持Web存储，还怎么保存？）
  if (localStorage) {

    //询问是否保存位置
    if (confirm(
  "Do you want to save your current position in the maze, for next time?")) {
```

```
      //保存两个坐标值
      localStorage.setItem("mazeGame_currentX", x);
      localStorage.setItem("mazeGame_currentY", y);
    }
  }
}
```

提示 建议你也像这个例子一样选择长键名，如mazeGame_currentX。因为必须保证键名的唯一性，这样网站的两个页面就不会意外地使用同一个键保存不同的数据。在只有一个存储空间的系统中，很容易发生命名冲突，而这也正是Web存储的一个弱点。为了避免这个问题，最好提前做个规划，选择一种有逻辑性、能自解释的键名命名方案。例如，假设另一个页面也有一个迷宫游戏，那么可以考虑将页面名也放到键名中，比如Maze01_currentX。

这个例子展示了如何保存应用状态（当前的位置）。如果你不想让用户每次离开游戏页面都看到同样的提示消息，可以添加一个“自动保存位置”复选框。然后，只要用户勾选复选框就保存位置信息。当然，你还得多保存一个复选框的值，而这也就是保存应用偏好的例子了。

这样，当下次再打开迷宫游戏页面时，就可以检查是否有之前保存的位置：

```
//支持本地存储功能吗?
if (localStorage) {
  //取得数据
  var savedX = localStorage.getItem("mazeGame_currentX");
  var savedY = localStorage.getItem("mazeGame_currentY");

  //如果变量为null，则说明没有保存的数据
  //否则，用保存的数据设置新坐标
  if (savedX != null) x = Number(savedX);
  if (savedY != null) y = Number(savedY);
}
```

这个例子使用了JavaScript的Number()函数，用于把保存的数据转换为有效的数值。10.2.3节会进一步介绍这样做的必要性。

10.1.3 浏览器对Web存储的支持情况

Web存储是现代浏览器支持情况最好的HTML5功能之一 。你能找到的不支持Web存储的可能就只有IE7了，好在它就要消失在历史的长河中了。

如果需要兼容IE7，可以用cookie来模拟Web存储。虽然不完美，但却可行。虽然没有官方的脚本可以帮你实现模拟，但在GitHub页面http://tinyurl.com/polyfill的“Web Storage”部分，还是可以找到不少有用的东西。

我们将在10.2.5节介绍onStorage事件，这个Web存储功能没有得到广泛的支持。尤其是IE8，支持Web存储却不支持onStorage事件（IE9及其之后的版本修复了这个问题，全面支持Web存储）。可以用onStorage添加一个不太重要的功能，但最好不要。

10.2　深入 Web 存储

现在我们已经掌握了Web存储的基础知识，包括保存和读取数据。可是，在学以致用之前，还有一些重要的知识点以及有用的技术需要再掌握一下。接下来的几节会介绍怎么从Web存储中删除数据项，怎么检索当前保存的所有数据。另外，还会介绍如何处理不同的数据类型、保存自定义对象和响应存储数据变化。

10.2.1　删除数据项

这个任务已经简单得不可能再简单了。只要调用removeItem()方法，传入键名，就可以删除不想要的数据项：

```
localStorage.removeItem("user_name");
```

要不然，就调用更厉害的clear()方法，清空网站在本地保存的会话数据：

```
sessionStorage.clear();
```

10.2.2　查找所有数据项

要搜索某一个数据项，只要知道键名即可。但这里会再告诉你一个更有意思的技巧：不必知道任何键名，使用key()方法从本地或会话存储中取得（当前网站保存的）所有数据项。这个技巧非常适合调试排错。当然如果你想知道其他页面都保存了哪些数据，或者都使用了什么样的键名，也可以使用它。

图10-3展示了一个实际使用这个技巧的例子。

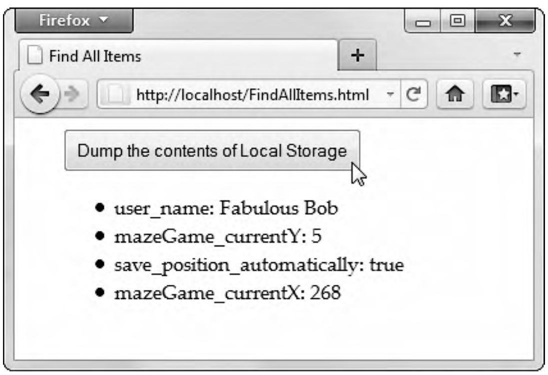


图10-3：单击按钮，页面中就会列出本地存储中的数据

在这个例子中，单击按钮会执行findAllItems()函数，该函数会遍历本地存储中的所有数据项，其代码如下：

```
function findAllItems() {
```

```
    //取得用于保存数据项的<ul>元素
    var itemList = document.getElementById("itemList");

    //清除列表
    itemList.innerHTML = "";

    //遍历所有数据项
    for (var i=0; i<localStorage.length; i++) {
      //取得当前位置数据项的键
      var key = localStorage.key(i);

      //取得以该键保存的数据值
      var item = localStorage.getItem(key);

      //用以上数据创建一个列表项
      //添加到页面中
      var newItem = document.createElement("li");
      newItem.innerHTML = key + ": " + item;
      itemList.appendChild(newItem);
    }
  }
```

10.2.3　保存数值和日期

到目前为止，我们还遗漏了关于Web存储的一个重要细节。那就是在通过localStorage和sessionStorage保存数据时，该数据会自动被转换为文本字符串。对于本来就是文本的数据（如在文本框中输入的用户名），这当然没问题。但数值就不一样了。比如10.1.2节的那个例子，保存的就是迷宫当前的位置信息。如果忘了把文本转换成数值，那么就会碰到下面所示的问题：

```
//取得x坐标
//假设保存为文本"35"
x = localStorage.getItem("mazeGame_currentX");

//给坐标加上一个数值
//然而，JavaScript会把"35"+"5"转换成"355"
x += 5;
```

不幸的是，代码的运行结果并不是我们想要的。因为x是个字符串，JavaScript也把5转化为字符串了，所以实际运行的是"35"+ "5"，结果为355，而不是35+5。事实上，这段代码会让小笑脸跳到完全错误的位置，或者跳出迷宫。

问题在于，JavaScript认为你是想把两段文本拼起来，而不是要执行数学计算。要解决这个问题，就需要给JavaScript一个提示，告诉它你想计算两个数值的加法。办法有很多，但使用Number()函数就很好：

```
x = Number(localStorage.getItem("mazeGame_currentX"));

//这样，JavaScript就可以正确地计算35+5，返回40
x += 10;
```

文本和数值还算容易处理的。如果你想在Web存储中保存其他类型的数据，就要多加留意。有些数据类型有方便的转换方法。比如，像下面这样保存日期：

```
var today = new Date();
```

结果并不会保存日期对象，而是会保存一个文本字符串。比如“Sat Jun 07 2014 13:30:46”。可是，要把这样的文本转换回日期对象可不容易。若没有日期对象，也就不能以相同方式来操作日期，比如不能调用日期对象的方法执行日期计算。

为解决这个问题，可以先按照既定的格式把日期转换成相应的文本，然后再根据取得的文本创建日期对象。下面就是一个例子：

```
//创建日期对象
var today = new Date();

//按照YYYY/MM/DD的标准格式把日期转换成文本字符串
//然后保存为文本
var todayString = today.getFullYear() + "/" +
 today.getMonth() + "/" + today.getDate();
sessionStorage.setItem("session_started", todayString);

...

//取得日期文本，并基于该文本创建新的日期对象
//这是因为文本格式是有效的日期形式
today = new Date(sessionStorage.getItem("session_started"));

//使用日期对象的方法，比如getFullYear()
alert(today.getFullYear());
```

运行以上代码，会弹出一个显示年份的消息框，也就说明你重新创建了日期对象。

10.2.4 保存对象

上一节介绍了在Web存储中保存数值和日期时会把它们转换成文本，而将来使用时还要再转换回去。这些转换都有JavaScript函数的帮助，首先是Number()函数，然后是文本到日期转换时的一些技巧，依靠日期对象固有的方法。然而，还有很多其他对象不能这样转换，尤其是自定义对象。

比如9.2.4节的人格测试，那个例子使用了两个页面。测试者在第一个页面回答问题然后得到一个分数，而第二页会显示测试结果。当时为在两个页面间传递数据，使用了嵌入在URL中的查询字符串参数。这是传统HTML中的做法（当然使用传统的cookie也可以）。但有了HTML5，利用本地存储共享数据才是最佳方案。

可是也有一个难题。测试数据包含5个数据，分别对应五个人格因素。当然，可以分别保存这5个数值，但要是能把所有人格因素保存到一个自定义对象中，岂不更简单明了？为此我们可以定义一个PersonalityScore对象：

```
function PersonalityScore(o, c, e, a, n) {
  this.openness = o;
  this.conscientiousness = c;
```

```
    this.extraversion = e;
    this.agreeableness = a;
    this.neuroticism = n;
  }
```

定义了PersonalityScore对象之后，只要1个数据项（而非5个）就可以保存所有数据。（关于自定义对象，参见附录B。）

为了把自定义对象保存到Web存储中，必须先把对象转换成文本形式。要自己写转换代码，那麻烦可就大了。好在JavaScript有一个更简单的、标准化的机制叫JSON编码。

JSON（JavaScript Object Notation，JavaScript对象表示法）是把结构化数据——类似封装在对象中的那些值——转换为文本的一种简便格式。而且浏览器原生支持JSON编码。也就是说，直接调用JSON.stringify()，就可以把任何对象连同其数据转换为文本形式。调用JSON.parse()则可以把文本转换回对象。以下就是在测试中转换PersonalityScore对象的代码。在测试者提交答案时，页面会计算分数（但不显示），创建对象，保存它，然后再打开新页面：

```
//创建PersonalityScore对象
var score = new PersonalityScore(o, c, e, a, n);

//将其保存为方便的JSON格式
sessionStorage.setItem("personalityScore", JSON.stringify(score));

//转到结果页
window.location = "PersonalityTest_Score.html";
```

到新页面后，再从会话存储中取出JSON文本，使用JSON.parse()方法将其转换回对象。以下就是相应的代码：

```
//JSON文本转换为原来的对象
var score = JSON.parse(sessionStorage.getItem("personalityScore"));

//从对象中取得数据
lblScoreInfo.innerHTML = "Your extraversion score is " + score.extraversion;
```

要查看这个例子的完整代码，包括每个人格因素的计算过程，请参考http://prosetech.com/html5中的页面。要了解更多关于JSON的内容，看看JSON格式的数据长什么样，请参考http://en.wikipedia.org/wiki/JSON。

10.2.5 响应存储变化

Web存储也为我们提供了在不同浏览器窗口间通信的机制。具体来说，就是在本地存储或会话存储发生变化时，其他查看同一页面或者同一站点中其他页面的窗口就会触发window.onStorage事件。因此，如果你在www.GoatsCanFloat.org/storeStuff.html页面中改变了本地存储，那么打开www.GoatsCanFloat.org/checkStorage.html页面的窗口会触发onStorage事件。（当然，必须是同一台计算机中相同的浏览器打开的页面，这一点你已经知道了。）

所谓存储变化，指的就是向存储中添加新数据项，修改既有数据项，删除数据项或清除所有

数据。但是，那些对存储不产生任何影响的操作（比如用既有的键名保存相同的值，或者清除原本就是空的存储空间），不会引发onStorage事件。

下面看看图10-4所示的页面。可以在这个页面中向本地存储添加任何数据项，只要在相应的文本框中输入键和值即可。保存新数据项时，第二个页面就会报告保存了什么。

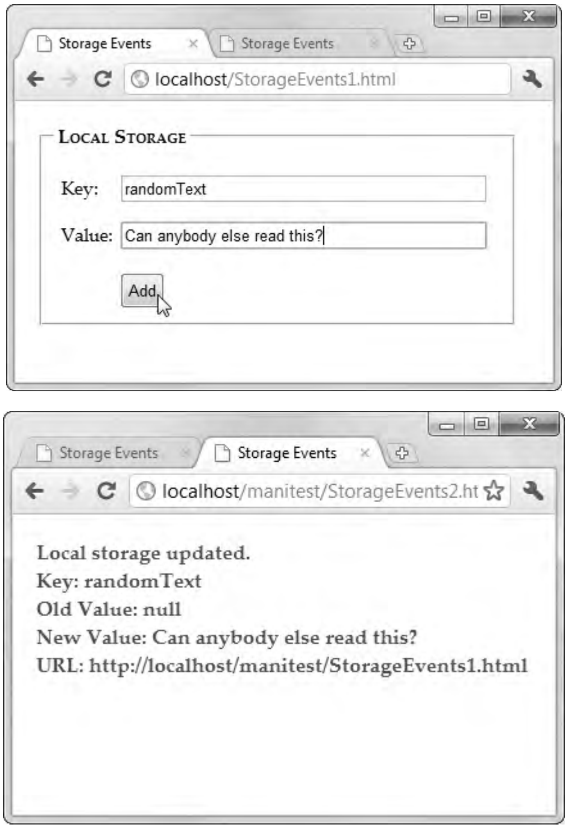


图10-4：为了体验onStorage事件，同时打开StorageEvents1.html和StorageEvents2.html。当你在第一个页面（上）中添加或修改数据项时，第二个页面（下）会响应该事件，并报告结果（底部）

图10-4所示的示例涉及两个页面，第一个页面负责保存数据。在这个页面中，单击Add按钮会触发一个小函数addValue()，其代码如下：

```
function addValue() {
  //取得两个文本框中的值
  var key = document.getElementById("key").value;
  var item = document.getElementById("item").value;

  //在本地存储中保存数据项
  //（如果同名键已经存在，则用新值替换旧值）
  localStorage.setItem(key, item);
}
```

第二个页面很简单，就是在页面加载后为window.onStorage事件添加一个处理函数，代码如下：

```
window.onload = function() {
  //把onStorage事件与storageChanged()函数联系起来
  window.addEventListener("storage", storageChanged, false);
};
```

以上代码与我们前面展示的添加事件处理程序的代码有所不同。在此，我们没有设置window.onstorage事件，而是调用了window.addEventListener()。这是为了确保代码在所有浏览器中都能运行，而这样写是最简单的形式。如果直接设置window.onstorage事件，那么这个例子在Firefox中就不能运行（因为Firefox的window对象没有onstorage属性）。

注意 Web老手可能还记得addEventListener()方法不能在IE8（或更早版本中）使用。在这个例子中，其实不用考虑这个问题，因为IE8根本就不支持存储事件。

storageChanged()函数的任务很简单，只是取得更新的信息，然后通过页面中的<div>元素显示出来：

```
function storageChanged(e) {
  var message = document.getElementById("updateMessage");
  message.innerHTML = "Local storage updated.";
  message.innerHTML += "<br>Key: " + e.key;
  message.innerHTML += "<br>Old Value: " + e.oldValue;
  message.innerHTML += "<br>New Value: " + e.newValue;
  message.innerHTML += "<br>URL: " + e.url;
}
```

可见，onStorage事件提供了不少信息，包括发生变化的键和值、原来的值（oldValue）、新值（newValue）和导致此次变化的页面URL。如果onStorage事件反映的是插入新数据项，那么e.oldValue属性要么是null（在大多数浏览器中）或者空字符串（在Internet Explorer中）。

注意 如果同时打开了同一站点的多个页面，那么这些页面会依次发生onStorage事件，只有导致变化的页面（即前面例子中的StorageEvents1.html）不会发生该事件。不过，IE是个例外，它也会在导致变化的页面触发onStorage事件。

10

10.3 读取文件

作为HTML5的一部分，Web存储得到了很好的支持。但这并不是存取数据的唯一方式。为了实现与存储相关的不同任务，也出现了其他几种不同的标准。其中一个就是File API，从技术角度讲，它并不是HTML5规范的内容，但得到了现代浏览器较好的支持，IE除外（IE10才开始支持）。

File API，这名字听起来蛮大气嘛！不知道的，还以为它是针对浏览器读写硬盘文件而制定的一个全方位的标准。然而，它可没有那么高远的志向，或者说没有那么强大。简单地说，File API只是规定怎么从硬盘上提取文件，直接交给在网页中运行的JavaScript代码。然后代码可以打开文件探究数据，无论是文本文件还是其他文件。注意，关键在于文件会被直接交给JavaScript代码。与以往的文件上传不一样，File API不是为了向服务器提交文件设计的。

另外，关于File API不能做什么，也非常值得注意。很明显，它不能修改文件，也不能创建新文件。想保存任何数据，你都要采用其他办法，比如通过`XMLHttpRequest`（参见12.1.1节）把数据发送到服务器，或者把它保存在本地存储空间中。

说到这儿，有读者可能会认为File API不如本地存储有用。嗯，对于大多数站点而言，的确如此。可是，从某种意义上讲，File API却为HTML扩展了疆界，至少在没有插件的情况下，通过它能够走得更远。

注意 目前，File API对于某些专门网站是不可或缺的。将来，随着其功能的增强，还会变得越来越重要。比如，将来的某个版本可能会允许网页在本地硬盘上写文件，让用户通过“保存对话框”控制文件名和保存位置。Flash浏览器插件已经具备了这种能力。

10.3.1 取得文件

在通过File API操作文件之前，首先必须取得文件。为此，有三种方式可以选择；实际上，归根结底只有一种方式，那就是必须由访客自己选择文件然后提交给你。

这三种方式如下。

- **使用`<input>`元素**。将其`type`属性设置为`file`，这样就能得到一个标准的上传文件框。不过，编写一点JavaScript来利用File API，就可以在本地打开文件。
- **隐藏的`<input>`元素**。嫌`<input>`元素太难看？为了保证风格一致，可以把`<input>`元素隐藏起来，显示一个漂亮的按钮。用户单击按钮，就通过JavaScript调用隐藏的`<input>`元素的`click()`方法。这样就会显示标准的文件选择对话框。
- **拖放**。如果浏览器支持拖放，可以从桌面或资源管理器中把文件拖放到网页上。

接下来几节就讨论这几种方式 。

10.3.2 用`<input>`读取文本文件

使用File API可以直接读取文本文件的内容。图10-5展示了一个例子，是一个页面读取了一个网页文件中的标准，然后显示出来。

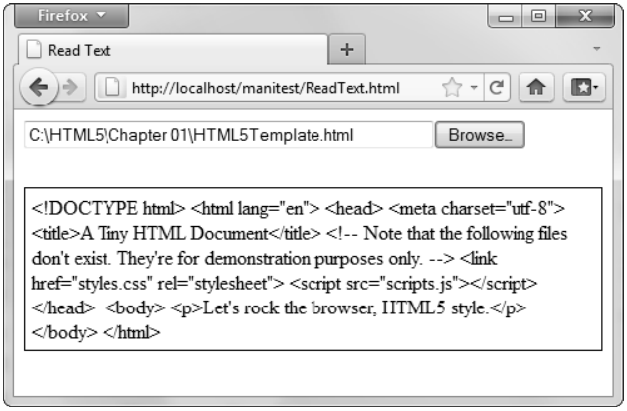


图10-5：单击Browse按钮（或Choose File，在Chrome中），选择一个文件，然后单击OK。无需上传，网页中的JavaScript就能取得文本文件，把内容复制到页面中

要创建这个例子，首先要使用一个`<input type="file">`元素，这样就能得到文本框和浏览器按钮：

```
<input id="fileInput" type="file" onchange="processFiles(this.files)">
```

不过，与往常属于`<form>`元素并且会将文件发送给Web服务器的`<input>`元素不同，这个`<input>`有自己处理文件的方式。访客选择了一个文件后，就会触发这个`<input>`元素的onChange事件，因而就会执行processFiles()函数。这个函数将会通过JavaScript来打开文件。

下面我们就来一行一行地分析processFiles()函数。这个函数首先必须从`<input>`元素提供的文件集合中取得第一个文件。除非你允许用户选择多个文件（使用multiple属性），否则文件集合中只会有一个文件，而该文件在集合中的索引就是0：

```
function processFiles(files) {
  var file = files[0];
```

注意 每个文件对象都有三个有用的属性：name属性保存文件名（不包含路径），size属性保存文件的字节大小，而（如果可以确定的话）type属性保存文件的MIME类型（参见5.3.1节）。可以分别读取这三个属性，然后加入判断，比如拒绝处理超过一定大小的文件，或者只允许某种类型的文件。

然后，创建FileReader对象，以便处理文件：

```
  var reader = new FileReader();
```

紧接着，差不多就可以调用FileReader的方法来提取文件内容了。但这个对象的方法都是异步的，也就是说可以不必等待数据而立即读取。要取得文件内容，首先要处理onLoad事件：

```
  reader.onload = function (e) {
    //这个事件发生，意味着数据准备好了
```

```
    //把它复制到页面的<div>元素中
    var output = document.getElementById("fileOutput");
    output.textContent = e.target.result;
  };
```

最后，在这个事件处理程序之后，调用FileReader的readAsText()方法：

```
  reader.readAsText(file);
}
```

这个方法会把文件内容转换成一个长字符串，保存在发送给onLoad事件的e.target.result中。

readAsText()方法只能处理包含文本内容（而不是二进制内容）的文件。HTML文件当然没有问题，图10-5展示的就是读取HTML文件的结果。CSV格式也是各种有用的纯文本格式中的一种，它是所有电子表格程序都支持的一种导出格式。XML也是纯文本格式，它是程序间交换数据的一种标准。（XML也是Office XML格式的基础，因此可以使用readAsText()方法直接处理.docx和.xlsx文件。）

注意 JavaScript语言还有内置的XML解释器，因此可以从XML文件中直接提取所需内容。当然，处理XML文件要编写很多代码，而且处理大文件的效率也不高。与把大文件上传到Web服务器并在服务器上处理相比，在本地处理大XML文件没有多大优势。不过，我们至少能够看到File API带来了哪些可能性，而这些功能是几年前的人们连想也不敢想的。

---

readAsText()只是众多读取文件的方法之一。FileReader对象提供的方法还有：readAsBinaryString()、readAsDataURL()和readAsArrayBuffer()。

其中，readAsBinaryString()方法可以让应用处理二进制编码的数据，但基本上就是把数据保存到一个文本字符串中，效率不高。如果你真想要解释二进制数据，恐怕就要处理好特别复杂的编码问题。

对那些需要自己认真地做数据处理的开发者来说，readAsArrayBuffer()是更好的选择。这个方法将数据读到一个数组中（参见附录B），每个数组项代表一字节数据。这套方案的优势是可以用来创建Blob对象，然后切分成更小的二进制数据块，以便逐块处理。

注意 Blob是binary large object的缩写，意思是一大块数据。Blob是File API的优势；要了解更多相关知识，可以参阅Mozilla持续更新的文档：http://tinyurl.com/file-blob。

而readAsDataURL()方法则让我们能方便地取得图片数据。10.3.5节将介绍如何使用这个方法。不过，我们该花点时间，先把前面的例子页面改造得漂亮些。

10.3.3 替换标准的上传控件

Web开发人员一致认为，用于提交文件的标准<input>控件非常难看。虽然必须得用它，但可以不让任何人看见它。换句话说，就是像下面这样把它隐藏起来：

```
#fileInput {
  display: none;
}
```

接着再添加一个新的按钮，用于触发提交操作。一个普通的按钮就可以胜任，而且可以任意修改其外观：

```
<button onclick="showFileInput()">Analyze a File</button>
```

最后一步是处理按钮单击事件，通过该事件来手工调用隐藏的<input>元素的click()方法：

```
function showFileInput() {
  var fileInput = document.getElementById("fileInput");
  fileInput.click();
}
```

这样，单击按钮就可以运行showFileInput()函数，该函数会模拟单击隐藏的Browse按钮，并显示出对话框供访客选择文件。访客选择了文件后，又会触发隐藏的<input>元素的onChange事件，于是processFiles()函数运行，一切跟以前一样。

10.3.4 一次读取多个文件

没有理由限制用户一次只能提交一个文件。HTML5也支持一次提交多个文件，只要为<input>元素添加multiple属性即可：

```
<input id="fileInput" type="file" onchange="processFiles(this.files)"
multiple>
```

这样，用户就可以在打开的对话框中一次选择多个文件了（比如在Windows中按Ctrl键并单击多个文件，或者用鼠标拖出一个选择框）。支持选择多个文件，代码也要相应修改。换句话说，不能再像前面例子中那样，只取得集合中的第一个文件了。这次，要使用for循环来依次处理每个文件：

```
for (var i=0; i<files.length; i++) {
  //取得下一个文件
  var file = files[i];

  //为这个文件创建FileReader对象，然后运行相同的代码
  var reader = new FileReader();
  reader.onload = function (e) {
    ...
  };
  reader.readAsText(file);
}
```

10.3.5 通过拖曳读取图片文件

前面我们看到了，FileReader处理文本内容只需要一步。同样，处理图片内容也这么简单，而这就要归功于readAsDataURL()方法。

图10-6展示了一个涉及两项功能的例子：处理图片和文件拖放。提交的图片文件用于绘制元素的背景。当然也可以把图片绘制到画布中，然后利用画布的原始像素处理功能来修改图片（参见9.5.3节）。综合利用该技术，可以让用户把图片拖到页面中，然后在图片上绘制或者修改图片，最后再使用XMLHttpRequest调用（参见12.1.1节）把结果上传到服务器。



图10-6：为页面提供图片有两种方式：使用下面的文件上传控件，或者把一或多个图片拖放到虚线框中

要创建这个页面，首先要添加一个元素，以捕获拖放过来的文件。在我们的例子中，这个元素是一个名为dropBox的<div>：

```
<div id="dropBox">
  <div>Drop your image here...</div>
</div>
```

写几行简单的样式声明，就可以为其设置好指定大小、边框和颜色：

```
#dropBox {
  margin: 15px;
  width: 300px;
  height: 300px;
  border: 5px dashed gray;
  border-radius: 8px;
  background: lightyellow;
  background-size: 100%;
  background-repeat: no-repeat;
  text-align: center;
}

#dropBox div {
  margin: 100px 70px;
  color: orange;
  font-size: 25px;
```

```
  font-family: Verdana, Arial, sans-serif;
}
```

眼光敏锐的读者肯定已经发现了background-size和background-repeat属性。这两个属性是为了接下来的功能作准备的。当把图片拖放到这个<div>上时，图片会作为它的背景。而background-size属性是为了缩小图片以全部显示，background-repeat属性则是为了不让图片重复显示。

为了处理放置文件的操作，需要处理三个事件：onDragEnter、onDragOver和onDrop。页面一加载完成，就会为这三个事件添加处理程序：

```
var dropBox;

window.onload = function() {
  dropBox = document.getElementById("dropBox");
  dropBox.ondragenter = ignoreDrag;
  dropBox.ondragover = ignoreDrag;
  dropBox.ondrop = drop;
};
```

其中，ignoreDrag()函数同时处理onDragEnter和onDragOver事件，前者在鼠标指针进入放置区时发生，后者在拖动文件的鼠标指针位于放置区之上时发生。之所以用同一个函数处理两个事件，原因就是不必对这两个事件作出反应，只要告诉浏览器自己什么也不做即可。这个函数的代码如下：

```
function ignoreDrag(e) {
  //因为我们在处理拖放，所以应该
  //确保没有其他元素会取得这个事件
  e.stopPropagation();
  e.preventDefault();
}
```

我们要响应的事件是onDrop，这个事件一发生，就说明要取得和处理文件了。不过，由于存在两种向页面提交文件的方式，所以drop()函数调用了实际上负责处理的processFiles()函数：

```
function drop(e) {
  //取消事件传播及默认行为
  e.stopPropagation();
  e.preventDefault();

  //取得拖进来的文件
  var data = e.dataTransfer;
  var files = data.files;
  //将其传给真正的处理文件的函数
  processFiles(files);
}
```

最后一个函数就是processFiles()，它会创建一个FileReader，为其onload事件添加一个函数，然后调用readAsDataURL()将图片转换为数据URL（参见8.2.3节）。

注意 正如在学习Canvas时所提到的，数据URL是一种用长字符串表示图片的方式。这种方式让传递图片数据变得十分方便。为了在网页中显示图片，可以将<img>元素的src属性设置为图片URL（正如9.1.1节所做的那样），也可以将CSS的background-image属性设置为图片URL（像这个例子中一样）。

```
function processFiles(files) {
  var file = files[0];

  //创建FileReader
  var reader = new FileReader();

  //告诉它在准备好数据URL之后做什么
  reader.onload = function (e) {
    //使用图像URL来绘制dropBox的背景
    dropBox.style.backgroundImage = "url('" + e.target.result + "')";
  };

  //读取图片
  reader.readAsDataURL(file);
}
```

FileReader还有其他事件，在读取图片文件的过程中可以选择使用。如果读取图片的时间比较长，可以通过onProgress事件（间歇性地触发）来确定已经加载了多大比例。（可以调用FileReader的abort()方法取消未完成的操作。）如果打开或读取文件时发生错误，会触发onError事件。而在操作完成时，则触发onLoadEnd事件（包括由于错误导致的终止）。

10.3.6　浏览器对File API的支持情况

File API得到了浏览器的支持，但不像Web存储那么可靠。表10-1展示了浏览器对它的支持情况。

表10-1　浏览器对File API的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android
最低版本	10	3.6	13	6	11.1	6	3

由于File API需要一些比普通网页更高的权限，所以通过JavaScript来实现浏览器尚未实现的功能是不现实的。因此，需要一个类似Flash或Silverlight的插件。比如，访问https://github.com/MrSwitch/dropfile，可以找到一个腻子脚本，利用Silverlight拦截拖放过来的文件，打开并将其内容交给网页中的JavaScript代码。

10.4　IndexDB：浏览器的数据库引擎

你可能熟悉数据库的概念——一种严格结构化的信息目录，可以往里塞任何类型的数据：人、产品、销售额，等等。许多网站都使用存储在Web服务器上的数据库。比如，当你在亚马逊上查

找一本书时，页面会从亚马逊巨大的数据库中提取详情。同样的情况还有在维基百科中查找一篇文章，在YouTube上查找一个视频，或者在百度上进行一次搜索。

数年来，故事一直是这样的：数据库是服务器的领地，没的商量！但HTML5带来了希望。何不让浏览器具备创建本地数据库（数据库保存在用户电脑上，而不是遥远的服务器上）的能力？一时间，突然出现各种方案，包括一度流行但现已废弃的Web SQL Databases标准。不久前，仍在发展中的IndexedDB标准一跃成为HTML5官方认可的本地数据库解决方案。

注意 “IndexedDB”这个名字强调了数据库的本质：用索引标识和排列数据。索引确保了你不会在两个不同的地方取到同一条数据，当你有海量数据时，索引还加快了数据读取速度。

服务器端和客户端数据库的区别

网站使用服务器端数据的方式有很多种，但都跟HTML5没关系。这是为什么呢？因为HTML5聚焦于加载到用户浏览器的标记语言代码、样式和JavaScript。你电脑上的浏览器并没有直接访问服务器数据库的方式。即便技术上是可能的，但会带来各种安全问题。因此，要和一个服务器端数据库交互，需要编写在服务器运行的代码。比如，网站通常使用PHP或者C#代码（当然还有许多其他技术）来完成服务器端数据库相关任务，比如生成一个检索结果页或者记录用户的购买行为。

在继续之前，先来搞清楚到底IndexedDB在Web应用中扮演什么角色，这很重要！因为众多原因，IndexedDB绝不会取代服务器端数据库。最重要的是，IndexedDB给每个用户创建一个独立的数据库，而使用服务器端数据的网站需要一个唯一的、集中分类的数据，可以分享给每个用户。

但是，IndexedDB在其他一些特定场景下很有用：

- **创建自给自足的离线应用**。在11章，我们将介绍如何利用HTML5缓存，让网页在即便没有网络连接的情况下也能运行，这是一项非常适合移动设备的技术。可以用IndexedDB让离线应用变得更能干。比如，页面可以在有网络连接的时候从服务器端数据库获取所需的数据（使用12.1.1节介绍的`XMLHttpRequest`对象），然后将数据保存到本地数据库，以便在离线时访问。
- **优化性能**。一些应用使用了大量的数据，如果持续地在需要时获取同样的数据，网页的性能会下降。类似地，如果创建一个通过复杂计算生成数据的应用，你一定不想浪费时间重复做同样的计算。解决方案是将所有需要的数据保存到一个IndexedDB数据库中。把它当成你的超级定制缓存吧。
- **改进本地存储**。正如我们前面了解到的，本地存储提供了一个地方来保存浏览器会话间的数据，并在页面之间传递。如果相对来说数据结构较简单并且不大，普通的JavaScript变量和对象就能胜任。但如果数据非常巨大或者结果非常复杂，用IndexedDB数据库来管理就会更加简单快捷。

注意 IndexedDB存储在几个重要方面表现得很像本地存储。最重要的是，一个IndexedDB数据库属于某个个人，使用特定的电脑和特定的浏览器，访问特定的网站。任何一项有变化（比如切换浏览器，用另一个账号登录，或者换用手机访问），网页都会对应一个新的IndexedDB数据库。

学习IndexedDB可能会让人怯步。首先，标准很复杂（一些苛刻的开发者用丑陋来形容）。Web开发者负责创建数据库，构建它的数据表，并且给它们填充数据。其次，IndexedDB使用异步模式，这意味着数据库任务在后台运行，不会阻塞代码运行或锁定页面。缺点就是这种更复杂的模式会让代码分散到不同地方。比如，开始一个数据库任务的代码和处理这个任务结果的代码不在一个地方。要理出这些代码块的执行顺序并且理解它们如何组合到一起，还是要花点精力的。

图10-7展示了一个数据库驱动的页面，我们将在本章随后分析它。它用IndexedDB功能的一些基本操作来保存链接信息。数据库中的每一条记录都由一个URL和一些相关信息组成，比如它的名字和说明。使用FavoriteSiteTracker.html页面，可以添加新的链接数据、浏览已有的数据，以及修改或删除某条数据——这些都是任何数据库代码的基本操作。

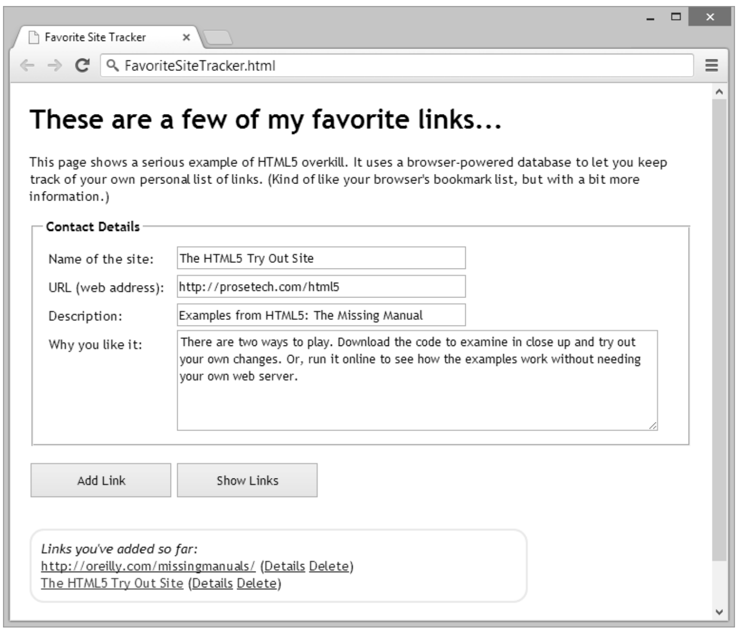


图10-7：这个例子用IndexedDB数据库保存链接信息。页面顶部有一个用来创建新的链接数据的表单。下面是一个数据库中已有链接的列表，显示了名称和URL（没有其他信息了）。如往常一样，可以访问http://prosetech.com/html5来体验这个例子

10.4.1 数据对象

在开始学习这个链接追踪示例之前，先来学习一下它所保存的数据——`LinkRecord`对象。传

统的数据库都是围绕数据表和字段，但IndexedDB简化了一下——存储对象。所以在创建数据库之前，最好先用JavaScript定义数据结构。

在这个例子中，数据库有一个唯一的数据表，里面的每条数据都是一个单个链接的信息。可以用一个对象定义函数将这些链接信息打包，代码如下：

```
function LinkRecord(name, url, description, notes) {
  this.name = name;
  this.url = url;
  this.description = description;
  this.notes = notes;
}
```

有了这段代码，就可以在需要时调用它来创建一个LinkRecord对象，包含描述性名称、URL、完整描述和注解。（如果对JavaScript对象创建有疑惑，请参阅附录B温习一下。）

10.4.2 创建并连接数据库

在任何使用IndexedDB的网页中，首要任务都是连接数据库。由于通常都是在页面第一次加载完成时开始处理这个任务，不妨就在window.onload事件回调中处理，下面是代码框架：

```
var database;

window.onload = function () {
  ...
}
```

database变量定义在函数外面，这样就可以在代码的任何地方访问到它。

所以，当页面加载完成时，代码需要做什么呢？这取决于你在这之前已经做了什么。

- 如果用户是第一次访问页面，则需要创建数据库并设置数据表 。
- 如果用户已经访问过，你只需要打开数据库并且设置database变量，这样它就能用了。

不论哪种方式，起始点都是window.indexedDB对象的open()函数。它有两个参数：要用（或创建）的数据库的名称，以及它的版本号。如果是一个全新的数据库，需设置版本号为1：

```
var request = window.indexedDB.open("LinksDB", 1);
```

注意 给数据库取什么名字都没关系，只要你保持一致并且没有在同一个网站的其他页面用过。

如前面所介绍的，IndexedDB的一切都是异步的。因此，当调用open()方法时，实际上并没有打开什么，而是发送了一个请求。这个请求是让浏览器在适当的时候打开数据库，并且最好是用一个后台线程，这样就不会阻塞页面，哪怕是一小会儿。

你会在几乎每一个IndexedDB任务中找到这种异步请求模式以及请求对象。要在任务完成时响应，需要至少设置两个事件处理器：一个处理成功结果的（onSuccess）和一个处理异常的（onError）。另外，使用open()时，还需要一个处理onUpgradeNeeded事件的处理器。

打开一个数据库时，能触发onSuccess事件表明指定版本的数据库存在。这时你需要设置数据库变量以便后面用到。你也可以开始读数据。在链接追踪器的例子中，有个单独的showLinks()函数在做这个事情（参见10.4.4节）。

```
request.onsuccess = function(event) {
  alert("Created or opened database successfully.");

  //让数据库可在任何地方访问
  database = request.result;
  //调用showLinks()函数读取数据库，并显示已存在的链接列表
  // list of links that already exist.
  showLinks();
};
```

onError事件同样简单。根据错误类型的不同，你可以进行纠正，或者简单地把错误报告给用户，用一个弹框或者直接显示在页面上。

```
request.onerror = function(event) {
  alert(request.error + " occurred.");
};
```

警告 不要犯忽略onError事件处理器的错误，否则，就不会有任何错误通知，你也不会有任何线索得知出现了错误。

onUpgradeNeeded事件是这3个里面最有趣的。它会在所请求的数据库版本不可用时触发。比如，如果请求的数据库版本是2而本地当前的数据库版本是1，就会触发这个事件。这个时候，可以升级数据库，比如添加一个新数据表。（可以从event.oldVersion参数里获取当前数据库版本。）onUpgradeNeeded事件也会在数据库根本不存在时触发，这种情况下浏览器会热情地创建一个空数据库，然后等你往里塞数据表。

要创建一个数据表，必须首先从request.result属性中获取数据库对象。然后，调用它的createObjectStore()方法。下面这段代码给数据库LinksDB创建了一个名为Links的数据表：

```
  request.onupgradeneeded = function(event) {
    var db = request.result;
    var objectStore = db.createObjectStore("Links", { keyPath: "url" });
  }
};
```

在传统的数据库编程中，创建数据表时需要指定字段名和数据类型。但IndexedDB用了一个更简单的、聚焦于对象的模式。创建数据表时，只需提供两个信息：数据表的名称和关键字路径，这是对象中作为主关键字的属性。

注意 关键字路径相当于数据对象中的一个属性。在这个例子中，url是小写的，以匹配数据对象中的属性名。翻回到10.4.1节的第一段代码，可以看到LinkRecord对象中定义了一个叫url的属性。

理解主关键字

主关键字是唯一标识每条数据的信息。在Links数据表例子中，每条链接数据有一个唯一的URL，因此用它做关键字说得过去 。这种机制可以防止意外地给同一个地址创建两条数据，也会强制浏览器给Links数据表创建索引，以便可以通过URL查找任何链接数据。

如果数据中没有明显适合做主关键字的属性，换句话说，没有一条可以确保唯一性的必要信息，那就可以添加数字ID号，这是一种历史悠久的数据库技术。更好的方式是，让浏览器在每次插入新数据时自动生成唯一的数字，这需要在创建数据库时设置autoIncrement属性为true，代码如下：

```
var objectStore =
db.createObjectStore("Links",
{ keyPath: "id", autoIncrement: true });
```

如果数据库需要多个数据表，可以用不同的数据对象，多次调用createObjectStore()。如果浏览器遇到问题，会触发request.onError事件。一旦数据表成功添加，就会触发request.onSuccess事件，代码就可以结束页面设置了。

注意 IndexedDB创建的数据库会永久保存，就像保存在本地存储的对象一样。但是，可以用window.indexedDB.deleteDatabase()方法删除一个数据库，一些浏览器还允许用户查看各个网站存储了多少数据并删除任何他不想要的。

10.4.3　在数据库中保存数据

数据库极客们用*数据操作*这个术语来描述在数据库中针对数据表数据的所有操作，无论是否包括读取、修改、插入或更新数据。在IndexedDB的世界里，数据操作同样也遵循最基本的4个步骤。一旦理解了这些模式，理解不同数据库任务的代码就容易了。

以下就是这些步骤。

(1) 创建一个事务。

任何时候要用IndexedDB做任何事情，不论是写数据还是读数据，都必须首先创建一个事务。在数据库中，事务是作为一个单元一起提交的一个或多个数据操作。如果事务中的任何一部分失败了，事务中的所有操作都会撤销，数据库会恢复到事务之前的状态。

事务对IndexedDB来说很重要，因为有许多会中断数据操作的状况。比如，当页面仍然在工作时，用户关闭了页面，浏览器会中断页面代码的执行。事务系统保障了即便发生了这种粗暴的打断，数据库仍然保持一致的状态。事实上，事务太重要了，没有它就不能对IndexedDB做任何操作。

注意 为了理解没有事务你会遇到的问题，想象一下你正在从一个银行账号向另一个账号转账。在事务处理机制下，第一步（从账号A转出金额）和第二步（把钱存入账号B）会一起成功或失败。但如果没有事务，可能会遭遇一个不那么愉快的状况： 从账号A转出金额成功了，但将金额转入账号B却失败了，导致了严重的现金丢失问题。

(2) 为事务获取对象存储。

对象存储是数据表的一个别名。因为IndexedDB数据表中的每一条数据实际上都是一个数据对象，所以这个名字合乎情理。

(3) 调用一个对象存储的方法。

对象存储是所有数据表操作功能的关键。比如，要添加一条数据，需调用它的`put()`方法。要删除一条数据，需调用它的`delete()`方法。方法会返回一个请求对象，进行下一步时必须用到这个对象。

(4) 处理成功和错误事件。

如你所知，事实上IndexedDB的一切都是异步的。如果想要在一个操作完成时做一些事，并且想要在问题恶化之前捕获错误，则需要处理`onSuccess`和`onError`事件，就像前文介绍的打开数据库所做的那样。

将这些步骤记在脑子里，来看看`addLink()`函数的代码，这个函数会在用户点击Add Link按钮时运行。

在用数据库之前，先要有数据。在链接追踪器的例子中，代码从表单中获取输入的数据，然后用它创建一个`LinkRecord`对象。这个任务是基本的JavaScript——很简单，找到元素，获取相应的值，然后使用10.4.1节第一段代码的`LinkRecord()`函数。

```
function addLink() {
  //从表单中获取数据
  var name = document.getElementById("name").value;
  var url = document.getElementById("url").value;
  var description = document.getElementById("description").value;
  var notes = document.getElementById("notes").value;
  //创建LinkRecord对象
  var linkRecord = new LinkRecord(name, url, description, notes);
```

现在，可以按照上文的第一步创建事务，调用`database.transaction()`方法并提供两个参数：

```
var transaction = database.transaction(["Links"], "readwrite");
```

第一个参数是所有需要被引入这个事务的数据表的列表，这个信息让IndexedDB可以锁定数据表，以阻止其他代码同时产生重叠或可能不一致的修改。技术上来讲，这个参数是一个包含了数据表名称的数组，这也是为什么它是用方括号括起来的。但在这个例子中，只有一个数据表被引入任务中。

第二个参数是一个指示事务类型的字符串。如果要创建一个可以对数据表进行任何操作的事务，不论是插入、更新还是删除数据，那就用`readwrite`；如果只需要读取数据，那就用`readonly`。

接下来是第二步，获取数据表所不可或缺的对象存储。可以简单地用transaction.objectStore()方法，只需提供数据表的名称，代码如下：

```
var objectStore = transaction.objectStore("Links");
```

要添加一条数据，使用对象存储的put()方法并传入数据对象：

```
var request = objectStore.put(linkRecord);
```

最后，需要处理onError和onSuccess事件来确定数据是否添加成功：

```
  request.onerror = function(event) {
    alert(request.error + " occurred.");
  };

  request.onsuccess = function(event) {
    //数据已添加
    //刷新显示（为了提高性能，可以只添加一个新项目，而不需更新整个列表）
    showLinks();
  };
}
```

注意 如果调用put()添加的数据与已存在的数据有相同的主关键字（比如，一个链接的URL已存在），浏览器会静默地用新数据替换已存在的数据。

10.4.4 在数据表中查询所有数据

查询是获取存储信息的基本工作。挑出一条数据，检索一组符合特定条件的数据，或遍历数据表中的所有数据，随你喜欢。

链接追踪器执行了一次完整的数据表浏览和一个单条数据的检索。它用数据表浏览创建了链接添加表单下方的链接列表。当点击列表中的Details链接时，它用数据检索来获取一个指定网站的详细信息，我们很快会介绍。

第一个任务更复杂一些。这是因为需要一个游标来浏览一个IndexedDB数据表。（数据库游标是一个追踪你在数据表中当前位置的对象，可以让你逐条访问数据。）

首先，创建一个事务并获取对象存储。这一次，不需要做任何修改，所以一个只读事务就够用了：

```
function showLinks() {
  var transaction = database.transaction(["Links"], "readonly");
  var objectStore = transaction.objectStore("Links");
```

接下来，用存储对象的openCursor()方法创建游标：

```
var request = objectStore.openCursor();
```

然后，给我们熟悉的onError和onSuccess事件添加回调处理器。onError事件处理器没什么特别的：

```
request.onerror = function(event) {
  alert(request.error + " occurred.");
};
```

onSuccess事件处理器更有趣一点，它需要逐步访问Links数据表中的数据。当它遍历时，会用链接列表创建一个HTML字符串。

```
//准备要插入页面的字符串
var markupToInsert = "";

request.onsuccess = function(event) {
  //创建一个游标
  var cursor = event.target.result;
```

作为初始值，游标定位在数据表的第一条数据，如果有的话。通过检测游标的值是true还是false来检测是否有数据。如果是true，则有可访问的数据。可以通过cursor.value属性访问到这条数据，代码如下：

```
if (cursor) {
  var linkRecord = cursor.value;
```

数据会以对象的形式返回。在链接追踪器的例子中，每条记录都是一个如假包换的LinkRecord对象，完整地包含了10.4.1节对象定义函数中的name、url、description以及notes属性。

获取到对象后，怎么处理它就由你决定了。在当前例子中，代码用LinkRecord数据构建了一个<a>元素。它用网站名称作为链接文本，用URL作为链接地址，代码如下：

```
markupToInsert += "<a href=" + linkRecord.url + ">" + linkRecord.name + "</a>";
```

现在，<a>保存在变量markupToInsert中。当代码完成了对数据库中每个LinkRecord的检查，并将它们的信息都添加到markupToInsert变量之后，就可以把这段HTML代码复制到页面中了。

在每一条链接后面添加两个可点击的文本之后，链接跟踪器变得更高级了。一个是“Detail”，另一个是“Delete”，可以在图10-7中看到（在屏幕的最底部）。

这些文本看起来像普通的链接，但实际上它们是<span>元素，并写死了点击时调用另外两个页面中的函数（getLinkDetails和deleteLink）。下面是创建这两个<span>的代码：

```
markupToInsert += "(" +
  "<span class='linkButton' data-url='" + linkRecord.url +
  "' onclick='getLinkDetails(this)'>Details</span>" + " " +
  "<span class='linkButton' data-url='" + linkRecord.url +
  "' onclick='deleteLink(this)'>Delete</span>" +
  ")<br>";
```

这里有个小技巧，Details和Delete触发器是<span>元素。简单起见，对应元素的URL用一个属性保存在每个<span>元素中。这样，当点击一个触发器时，代码可以迅速获取URL并用它在Links数据表中查找对应的数据。

注意 按照HTML5的惯例，保存URL的属性名称是data-url。data-前缀表明要用这个属性保存自定义数据，浏览器可以忽略。属性名剩下的部分可以是任何你想要的——这里，用url更合理，因为这个属性是用来保存URL的。

到目前为止，我们已经介绍了代码如何在检索时处理一个单条数据。当准备好访问下一条数据时，可以调用cursor.continue()方法。但是，不要试图处理更多数据，这是因为逐步访问数据是一个异步操作。当游标到达下一条数据时，onSuccess事件再一次触发，同样的代码又执行一次，这时可以为下一条数据添加HTML代码，如果它存在的话。

```
    cursor.continue();
  }
```

当到达数据表的最后一条数据时，游标的值会变为false。这时，就可以把HTML代码复制到页面中了：

```
      else {
        //如果一个结果也没有，用就用占位符文本代替
        if (markupToInsert == "") {
          markupToInsert = "<< No links. >>";
        }
        else {
          markupToInsert = "<i>Links you've added so far: </i><br>" +
            markupToInsert;
        }
        //插入代码
        var resultsElement = document.getElementById("links");
        resultsElement.innerHTML = markupToInsert;
      }
    };
  }
```

10.4.5 查询单条数据

在数据表中查询单条数据比获取全部数据要简单，因为不会被游标弄得一片混乱。因此，只需按照10.4.3节中介绍的固定4步程序，使用对象存储的get()方法。

如果点击链接追踪器例子中的一个"Details"链接，就会执行下面的代码。它获取对应的LinkRecord对象并解析这个对象的全部信息。

```
function getLinkDetails(element) {
  //从之前添加的data-url属性中获取这个链接的URL
  // earlier.
  var url = element.getAttribute("data-url");

  var transaction = database.transaction(["Links"], "readonly");
  var objectStore = transaction.objectStore("Links");

  //找到拥有这个URL的数据
  var request = objectStore.get(url);

  request.onerror = function(event) {
    alert(request.error + " occurred.");
  };

request.onsuccess = function(event) {
```

10

```
    var linkRecord = request.result;
    var resultsElement = document.getElementById("linkDetails");
    resultsElement.innerHTML = "<b>" + linkRecord.name + "</b><br>" +
      "<b>URL:</b> " + linkRecord.url + "<br>" +
      "<b>Description:</b> " + linkRecord.description + "<br>" +
      "<b>Notes:</b> " + linkRecord.notes;
  }
}
```

LinkRecord对象中的信息用来创建一个HTML代码片段，它随后会插入页面中链接列表下面的一个独立的容器中，图10-8展示了结果。

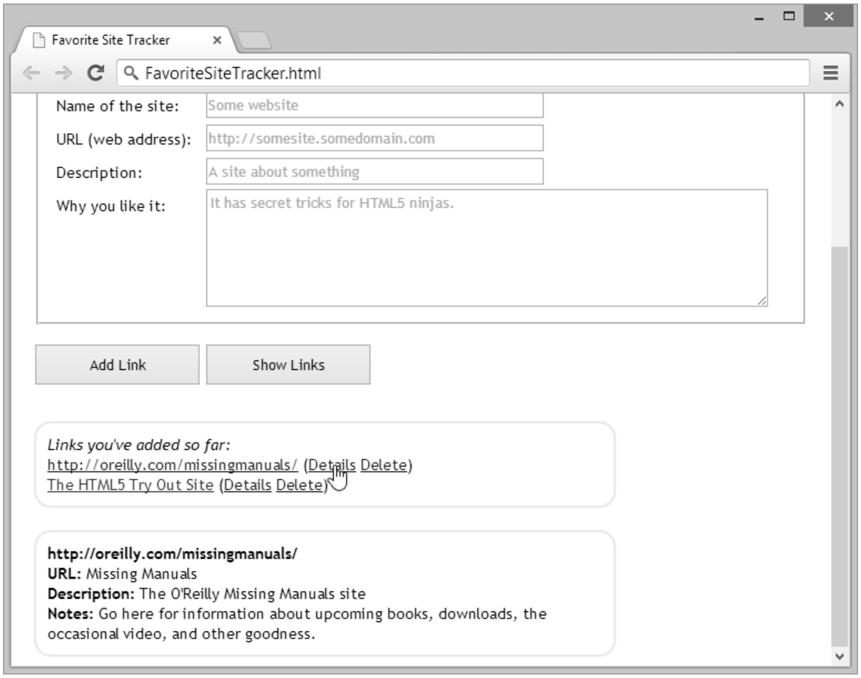


图10-8：点击Details时，代码会获取对应的数据并显示在底部容器中

10.4.6　删除一条数据

现在，你已经熟悉了支撑着所有数据操作的4个步骤。删除一条数据没有什么不同。很简单，用对象存储的delete()方法。

在链接追踪器例子中，“Delete”链接用来执行删除操作。点击之后，就会删除匹配这个URL的LinkRecord对象，代码如下：

```
function deleteLink(element) {
  var url = element.getAttribute("data-url");
```

```
    var transaction = database.transaction(["Links"], "readwrite");
    var objectStore = transaction.objectStore("Links");

    var request = objectStore.delete(url);

    request.onerror = function(event) {
      alert(request.error + " occurred.");
    };

    request.onsuccess = function (event) {
      //已删除数据
      //刷新显示
      showLinks();
    }
  }
```

我们已经介绍了IndexedDB中数据操作的主要方法，介绍了如何用对象存储添加或更新（用`put()`方法）以及删除（用`delete`）数据，还介绍了如何通过关键字的值获取单条数据（用`get`），或者浏览全部数据（用`openCursor`）。如果想要深入了解数据库，可以在http://tinyurl.com/objectstore查看Mozilla关于对象存储的文档，或者在www.w3.org/TR/IndexedDB上查阅详细的IndexedDB标准。

10.4.7 浏览器对IndexedDB的支持情况

IndexedDB是一个相对较新的标准，相对较新的浏览器才会支持。

表10-2展示了当前的支持情况。

表10-2 浏览器对IndexedDB的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android版Safari
最低版本	10	10	23	–	15	–	29

很遗憾，还没有桌面版或移动版的Safari支持IndexedDB。这个支持的空白是因为Safari的开发者们还在使用已废弃的Web SQL Database标准。如果需要Safari支持，可以用一个将IndexedDB操作转化为对应的Web SQL命令的腻子脚本（可以在http://tinyurl.com/DBpolyfill下载）。但如果想要一个不支持IndexedDB和Web SQL的浏览器支持IndexedDB，比如IE9或IE8，没戏！

第 11 章

离线应用

要访问网站，得先上网。这一点连三岁小孩都知道。那为什么还要用一章来讲离线应用呢？离线应用的想法看起来太不合适时宜了。毕竟，Web应用已经超越几代离线桌面应用，成为了世界的潮流。很多事，比如实时掌握卡戴珊的最新动态，或者订购一把办公椅，必须随时在线才有可能。不过别忘了，就算是Web应用也不可能永远不掉线。而在电脑短暂断网的情况下，它们应该能够正常工作。换句话说，一个实用的离线Web应用可以应付间歇性的网络中断。

对于使用智能手机和平板的用户来说，离线应用尤其重要。为了把问题说明白，让我们看一个例子。假设你在使用一个Web应用的时候恰好通过一条隧道。一进隧道，你就看到了一个错误页面，之前做过的工作全部丢失。等出了隧道之后，你必须全部从头来过。但是，如果这个Web应用支持离线功能，那么你就不会有这种痛苦的经历了。虽然部分功能可能会暂时不能用，但你不会被迫退出。（当然，有些隧道可能会很长，而真正做得好的离线Web应用能保证乘三个小时飞机不间断，如果你需要，甚至到刚果旅行三个星期都没问题。说到底，就是离线多长时间都不应该出问题。）

使用HTML5的离线应用功能，能把普通的网页转变成基于Web的“迷你应用”。如果再结合大量的JavaScript、第10章介绍的数据存储功能和第12章将要介绍的服务端通信功能，那这个“迷你应用”就几乎像智能手机和平板电脑上的原生应用一样自给自足，一样强大。

本章介绍怎么把网页（或一组网页）转换成离线应用。此外，还会介绍如何获悉网站可用或者网站离线，以便作出相应处理。

什么时候考虑离线

该不该让我的网页支持离线浏览呢？

离线应用并不适合所有网页。比如，把查询股票报价的网页转换成离线应用毫无意义，因为这个页面存在的唯一价值就是能够连接 Web 服务器更新数据。不过，如果是一个股票分析页面，那么下载一批数据后，即使离线也可以生成图表或分析报告。这样，在能上网时把数据下载下来，即使进了隧道也不妨碍你更改选项或单击按钮。

离线功能也适合那些具有交互性和**有状态**的网页，也就是动用大量 JavaScript 代码在内存中维护很多信息的网页。这些网页本身就可以实现很多功能，因此支持离线就有意义。不过，其中某个网页突然丢掉连接的代价也更大。想象一下，用户在执行一项复杂的任务时，突然中

断任务怎能不让人觉得讨厌呢？虽然包含简单内容的网页没多大必要做成离线的，但浏览器中的字处理工具显然需要离线支持。事实上，这样的离线应用恐怕恰恰能够取代功能更完善的桌面软件。

另外，还要考虑用户。如果有些用户不可能经常上网，或者可能会通过移动设备访问应用（比如为平板电脑设计的地图应用），那么支持离线功能就是必要的。如果情况并非如此，那么考虑离线只会自寻烦恼。

11.1 通过描述文件缓存资源

离线应用的一项基本技术就是缓存，即下载文件（如网页）并在用户计算机上保存一份副本。有了这份副本，即使计算机不能上网，浏览器也可以使用缓存的文件。当然，缓存不限于页面文件——样式文件、JavaScript文件、图片、字体文件以及任何页面执行时所需要的资源。

要创建一个离线应用，需要完成三步，下面是一个概述。

(1) 创建描述文件。

描述文件（manifest file）是一种特殊文件，告诉浏览器保存什么文件，不保存什么文件，以及用什么文件代替其他文件。描述文件中列出的所有需要缓存的内容，构成了所谓的*离线应用*。

(2) 修改网页，引用描述文件。

引用了描述文件，浏览器在请求页面时就会下载描述文件。

(3) 配置Web服务器。

这一步最重要，因为Web服务器必须以正确的MIME类型提供描述文件。稍后我们会介绍影响缓存的其他问题。

接下来的几节，我们就分别讲解这几个步骤。

传统缓存与离线应用

对Web开发而言，缓存并非新事物。浏览器经常利用缓存以避免下载相同的文件。毕竟，如果很多网页共用相同的样式表，那为什么每个页面都下载多次呢？不过，浏览器的这种缓存方式与离线应用的缓存方式可不一样。

触发浏览器中传统缓存的机制是Web服务器发送额外的信息，即cache-control**头部**，这个信息随同浏览器请求的文件一块发给浏览器。头部信息告诉浏览器是否应该缓存该文件，缓存多长时间再询问Web服务器该文件是否更新过。一般来说，缓存网页的时间比较短，而缓存网页资源（如样式表、图片和脚本）的时间比较长。

相对而言，离线应用由一个单独的文件（即描述文件）控制，也不限定时间。大致来说，它的规则是“如果网页是离线应用的一部分，如果浏览器已经缓存了该应用，如果应用的定义没有改变，那么就使用缓存的网页”。作为Web开发人员，可以声明一些例外，告诉浏览器不缓存某些文件，或者不用某个文件替代另一个文件。但是，不用考虑过期时间和其他一些烦琐的细节。

11.1.1 创建描述文件

描述文件是HTML5离线应用功能的关键所在。描述文件就是一个文本文件，其中列出了需要缓存的文件。

描述文件第一行一定是“CACHE MANIFEST”（全部大写）：

```
CACHE MANIFEST
```

然后，再列出需要缓存的文件。比如，下面的代码表示缓存两个网页（即9.2.4节的人格测试应用的两个页面）：

```
CACHE MANIFEST

PersonalityTest.html
PersonalityTest_Score.html
```

文件中的空格（包括上面的空行）是可选的，可以根据需要添加。

注意 别写错了。如果写了一个不存在的文件，浏览器会忽略整个描述文件。

为了离线应用的正常运行，浏览器必须缓存其所需的一切，包括网页和网页用到的资源（脚本、图片、样式表和嵌入的字体）。下面这个示例描述文件列出了所有相关的资源：

```
CACHE MANIFEST
# pages
PersonalityTest.html
PersonalityTest_Score.html

# styles & scripts
PersonalityTest.css
PersonalityTest.js

# pictures & fonts
Images/emotional_bear.jpg
Fonts/museo_slab_500-webfont.eot
Fonts/museo_slab_500-webfont.woff
Fonts/museo_slab_500-webfont.ttf
Fonts/museo_slab_500-webfont.svg
```

关于这个文件有两点要注意。首先，以#开头的行是注释，说明下面缓存哪些内容。其次，有些资源位于子目录下（比如emotional_bear.jpg，位于Images文件夹中）。只要这些目录与Web服务器中的目录对应，而且浏览器能够访问到该目录，就可以像这样把它们作为离线应用的一部分缓存下来。

复杂的网页一般都需要很多支持文件，因此描述文件也会很长很复杂。因此，最忌讳的问题是拼写错误，一个文件名写错，就会导致整个离线应用无法运行。不久的将来，Web编辑器等工具可以帮我们减少这种麻烦。它们可以根据选中的网页自动创建描述文件，同时为修改和维护描述文件提供辅助功能。

提示 有时候，可能不必缓存一些很大但又不重要的资源。比如，大照片或大的横幅广告。不过，假如缺少这些文件会影响页面呈现（比如导致错误消息、奇怪的页面空白或布局凌乱），那应该使用JavaScript在用户离线时调整页面，11.2.3节将介绍如何检测用户设置是否在线。

准备好描述文件后，可以把它保存在网站根目录下，与其他网页放在一起。描述文件的名字可以随便起，虽然应该用.appcache（比如PersonalityTest.appcache），但其他文件扩展名也可以（比如，HTML5诞生之初一些开发者用.manifest），而最新版的HTML5标准推荐用.appcache 。最重要的还是在Web服务器上进行配置，让它能够认识扩展名。如果你有权限设置Web服务器，可以根据11.1.3节的介绍完成配置。如果没有权限，可以问一问主机托管公司，看他们配置了什么扩展名来支持描述文件。

不要缓存有查询字符串的页面

查询字符串是URL尾部额外的一些信息，由一个问号分隔。通常，查询字符串用来在页面间传递信息。比如，人格测试例子原来的版本用查询字符串将人格测试得分从PersonalityTest.html传递到PersonalityTest_Score.html。当在第一个页面填完多选题并点击Get Score时，浏览器会跳转到像下面这样的一个URL：

```
http://prosetech.com/html5/PersonalityTest_
Score.html?e=-10&a=-5&c=10&n=5&o=20
```

那么问题来了，在HTML5缓存系统机制中，PersonalityTest_Score.html和PersonalityTest_Score.html?e=-10&a=-5&c=10&n=5&o=20是两个不同的请求。根据描述文件，第一个页面被缓存了。但第二个URL相当于指向了一个完全不同的页面，除非将页面名称和完整的查询字符串添加到描述文件中，否则它是不会被缓存的。因为不太可能把每种得分组合的URL都写进描述文件，所以也不太可能缓存带有查询字符串的PersonalityTest_Score.html页面。

要避免这个问题，不要同时使用缓存和查询字符串。比如，如果想要给人格测试例子添加缓存，可以用把得分存在本地存储的版本（11.1.1节所用的示例版本）。

11.1.2 使用描述文件

只是创建描述文件可不行，还必须让浏览器知道它在哪里。换句话说，你得在自己的网页中引用它。为此，要为`<html>`元素添加`manifest`属性，将该属性的值设置为描述文件的路径，比如：

```
<!DOCTYPE html>
<html lang="en" manifest="PersonalityTest.manifest">
...
```

而且，必须给离线应用包含的每个页面都添加同样的属性。在前面的例子中，也就是要修改两个文件：PersonalityTest.html和PersonalityTest_Score.html。

注意 一个网站可以有任意多个离线应用，每个应用分别有自己的描述文件即可。离线应用也可以使用相同的资源（比如样式表），但必须包含不同的网页。

11.1.3 把描述文件放到Web服务器

测试描述文件的时候需要一些耐心，任何微小的问题都可能导致静默失败，结束缓存过程。同样，在适当的时候也要试试看，看你的离线应用是否真能在不联网的情况下运行。

说到测试，从本地硬盘加载文件当然不行，必须得把应用上传到服务器（或者使用本机运行的测试服务器，比如Windows自带的IIS）。

测试离线应用的步骤如下。

(1) 确认已经配置好Web服务器，使用text/cache-manifest的MIME类型来解析描述文件（通常，这些文件的扩展名为.appcache）。

如果Web服务器以其他MIME类型（包括纯文本）交付描述文件，浏览器都会忽略描述文件。

注意 不同的Web服务器配置MIME类型的方式也不一样。如果你对这一块不熟悉，可以找服务器厂商咨询，或找一个熟悉Web服务器配置的朋友帮你设置MIME类型（第1步），同时修改缓存设置（第2步）。要了解有关MIME类型的更多信息，请参考5.3.1节，那里还用一个例子介绍了如何通过一个Web服务器账号添加新的MIME类型。

(2) 考虑在使用描述文件时关闭传统缓存机制（参见11.1节开头）。

这样做的原因是，Web服务器可能会告诉浏览器把描述文件缓存一段时间，就像告诉它缓存其他文件一样。这种做法无可厚非，但却可能给测试带来极大的难题。假设你后来又更新了描述文件，但浏览器仍然会使用缓存的旧描述文件，于是离线应用也将继续使用以前缓存的网页。（Firefox特别喜欢使用过时的描述文件，很讨厌。）为了避免这一点，应该配置Web服务器，让它告诉浏览器永远不要缓存描述文件。

还有，每个服务器软件都有自己的配置机制，但本质上还是告诉服务器收到.appcache文件的请求时返回一个无缓存的请求头。

(3) 在支持离线应用的浏览器中打开网页。事实上所有浏览器都支持，除了旧版的IE——你需要IE10及其更新的版本。

浏览器在打开使用了描述文件的网页时，可能会请求用户的许可，之后再下载文件。移动设备通常会请求用户的许可，因为设备本身的存储空间有限。桌面浏览器则不一定，比如Firefox会（如图11-1所示），而Chrome、IE和Safari不会。



图11-1：Firefox会在加载包含描述文件的网页时显示这条消息。单击Allow授权下载并缓存该描述文件中列出的文件。随后再浏览此网页，Firefox会检测描述文件是否有更新，如果有更新则自动下载新文件而不会再请求授权

用户同意之后（或者浏览器没有问），缓存过程就会开始。浏览器会下载描述文件，然后再下载描述文件中列出的所有文件。这个下载过程是在后台进行的，不会影响当前页面。就如同浏览器下载大图片或下载视频一样，同时会显示页面的其他部分。

(4) 模拟离线。

如果你测试的是远程服务器，那么断开网络连接。如果是在本地Web服务器（即运行在你的计算机中的服务器）中测试，停止网站（参见图11-2）。

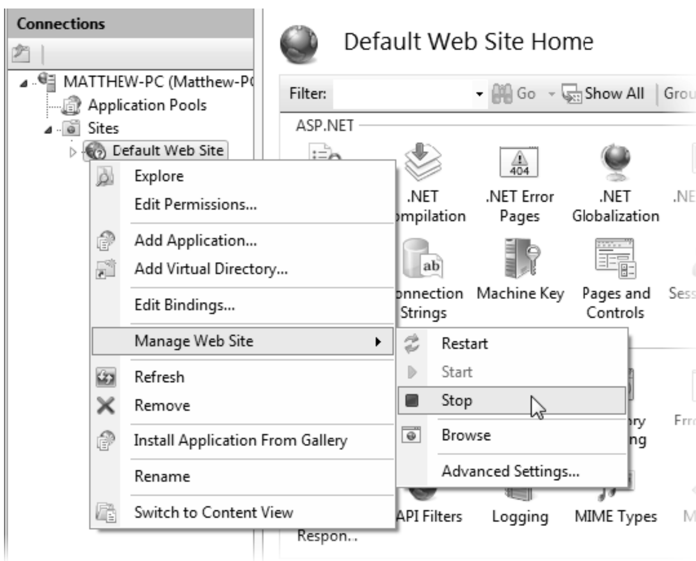


图11-2：具体如何停止网站取决于你使用的Web服务器软件类型。在Windows自带的IIS中（如图），在网站结点上单击右键，就可以找到相应的菜单项

(5) 浏览离线应用中的某个页面，然后刷新。

通常，在单击Refresh或Reload时，浏览器总是会尝试访问Web服务器。如果你请求的是一个普通页面，并且这时已经断开连接，那么请求会失败。可是，如果你请求的是离线应用中的一个页面，浏览器则会从缓存中找到该页面，悄悄地代替之前的页面。此时，单击链接可以自由跳转。如果你单击了不属于离线应用的页面，则会看到熟悉的“服务器没有响应”的错误消息。

我的离线应用离线不工作

离线应用功能还不是很稳定并且有点诡异，一个小小的错误就会导致它不能工作。如果你按照上面列出的步骤做了，但在尝试访问离线页面时还是看到了“服务器没有响应”的消息，可以试着排除以下常见的问题。

- **下载描述文件出了问题**。如果你没有把描述文件放在正确的位置，或者说浏览器没有找到它，那么出问题是自然的。但是，以正确的MIME类型（参见5.3.1节）来提供描述文件也同样重要。
- **下载描述文件中列出的文件出了问题**。比如，描述文件中包含一个不存在的图片。或者，要求浏览器下载Web字体文件，但该字体文件的类型又是Web服务器所不支持的。无论如何，只要浏览器下载一个文件时失败，它也会完全放弃（同时删除已经下载的所有数据）。为避免这个问题，先从简单的描述文件开始尝试，比如只包含一个网页，而不包含其他资源。如果不行，再查看一下Web服务器的日志，看浏览器到底请求了什么资源（这样就能知道是请求哪个文件出错导致了浏览器放弃）。
- **浏览器缓存了旧的描述文件**。浏览器有可能会缓存描述文件（根据传统的Web缓存规则），因此忽略更新的描述文件。如果你发现有些网页的确是被缓存的，但一些新网页却没有被缓存，那就可能是这个原因。解决方案是手工清空浏览器缓存（参见11.1.5节）。

11.1.4　更新描述文件

让应用离线工作是要解决的第一个难题。第二个难题是更新离线应用的内容。

就拿前面的例子来说吧，它缓存了两个页面。如果你更新了PersonalityTest.html，打开浏览器，重新加载这个页面，你看见的仍然是原先缓存的那个页面。无论你的计算机目前能否上网，都是如此。问题在于，只要浏览器缓存了应用，那么它就不会向Web服务器请求新内容。浏览器不管你是否更新了服务器上的页面，它只管用自己已经缓存的那个。由于离线应用没有过期一说，所以无论你过多长时间以后再看，就算是几个月以后再看，浏览器照旧还会忽略更新后的页面。

不过，浏览器会检测服务器上的描述文件是否有更新。因此，可以保存一份新的描述文件，把它放到服务器上，就可以解决这个问题了，对吧？

不一定。要触发浏览器更新缓存的应用，需要同时满足下列要求。

- **浏览器没有缓存描述文件**。如果浏览器在本地缓存了描述文件，它就不会再访问Web服务器去找新描述文件。在是否缓存描述文件这个问题上，不同浏览器有不同的处理方式。有的浏览器（比如Chrome）只要有条件就会检测服务器，看有没有更新的描述文件。但Firefox却遵循着传统的HTTP缓存规则，会将描述文件缓存一段时间。所以，为了减少麻烦，最好是让Web服务器明确告诉浏览器，不要缓存描述文件（参见11.1.3节）。
- **描述文件的保存日期必须是新的**。浏览器在检测服务器上的文件时，首先要看文件的最近更新时间。如果不是新保存的描述文件，那浏览器不会下载它。

- **描述文件中的内容要更新**。如果浏览器下载了新描述文件，结果却发现其内容没有变化，它同样会停止更新，而继续使用之前缓存的内容。这一点虽然不太合乎常理，但却很有价值。你想，重新下载一遍本来就已经缓存的内容，既耽误时间又浪费带宽，所以浏览器不在必要时是不会下载的。

如果你一直都在认真领会前面的内容，到这里可能会冒出一个问题来：要是描述文件确实没有什么好改的（因为你并没有添加任何文件），而我又想让浏览器更新缓存内容怎么办（因为原有的文件内容有变化）？这时候，你得稍微修改一下描述文件，让它看起来像是新的一样。为此，最好的办法就是添加注释，比如：

```
CACHE MANIFEST
# version 1.00.001
# pages
PersonalityTest.html
PersonalityTest_Score.html

# styles & scripts
PersonalityTest.css
PersonalityTest.js

# pictures & fonts
Images/emotional_bear.jpg
Fonts/museo_slab_500-webfont.eot
Fonts/museo_slab_500-webfont.woff
Fonts/museo_slab_500-webfont.ttf
Fonts/museo_slab_500-webfont.svg
```

等下一次再需要浏览器更新缓存时，只要把这个例子中的版本号改为1.00.002就行了。这样，既可以强制让浏览器更新已有内容，也可以记录自己更新的次数。

更新并不会瞬间完成。浏览器发现新描述文件后，会悄悄地下载所有文件，然后再用新下载的文件代替原来缓存的内容。下次用户再访问同一个页面（或刷新该页面），就会显示新内容。如果你想让用户马上就切换到新下载的内容，可以使用11.2.4节介绍的JavaScript技术。

注意 不能以只更新增量的方式更新离线应用。只要应用中有变化，浏览器就会抛弃所有旧文件，然后重新下载一遍，包括丝毫未改的那些文件。

清除浏览器缓存

测试离线应用时，手工清除浏览器缓存的作用很明显。这样，不必修改描述文件，也可以测试更新后的应用。

所有浏览器都提供了清除缓存的命令，但却把它们“藏”在了不同的地方。有的浏览器会记录每个离线应用使用的空间（参见图11-3）。这些信息让你可以判断哪个应用缓存失败了，比如没有列出来的或者缓存大小没有预期那么大的。当然，这样也可以一个应用一个应用地删除缓存文件，做到互不影响。



图11-3：上：在Firefox中，选择Options ➤ Advanced，然后选择Network选项卡。可以看到每个网站使用的缓存空间，也可以清除任何网站的缓存——选择该网站，再单击Remove即可。图中显示目前只缓存了一个站点，这个站点的域是localhost（也就是当前计算机的测试服务器）。下：要在Chrome中看到类似的信息，可以在地址栏中输入：chrome://appcache-internals

11.1.5 浏览器对离线应用的支持情况

相信大家都已经知道了，除了拖HTML5后腿的IE，所有主流浏览器都支持离线应用。有的浏览器很早就开始支持离线应用了，目前可以确定有Firefox、Chrome和Safari。但是IE直到IE10才开始支持，这意味着当下流行的IE9和IE8没有缓存。可是，不同浏览器支持离线应用的方式却不大一样。最重要的不一样是它们分配给离线应用的存储空间。这个差别非常重要，因为它决定了哪些网站可以做成离线应用，而哪些不能（参见10.1.5节）。

有没有让不支持离线应用的浏览器（如IE9）支持它的办法呢？没有什么值得尝试的好办法。不过，这并不妨碍你使用离线应用功能。毕竟，离线应用只是一个补充而已。

不支持离线功能的浏览器照样可以访问你的网站，只要能上网就行。对于那些需要离线浏览的人，比如经常出差的人，他们自己会找一个支持离线功能的浏览器，以备没有网络时使用。

缓存空间有多大

缓存有没有限制？

不同浏览器对离线应用缓存的限制有很大不同。

移动浏览器是最明显的例子。因为移动设备本身空间有限,所以对缓存的限制也比较严苛。桌面浏览器慷慨许多，但这也意味着不可预知。浏览器可能给每个网站分配一个固定的空间限制，也可能基于当前本地的剩余空间和一些其他因素计算一个合适的配额。通常，空间是由几个HTML5功能共享的，比如，如果使用了File API或IndexedDB功能（参见第10章），浏览器可能会给这些功能以及应用缓存分配一组空间。

显然，浏览器间的这种不一致性会导致问题。如果一个离线应用试图超额使用缓存，浏览器会静静地放弃并扔掉所有已下载的文件。不仅是你浪费了时间和空间，你的用户也没捞着离线的好。他们不得不在线使用你的应用。

最佳经验法则是给苹果设备（像iPad和iPhone）分配50 MB空间，给Android设备分配将近85 MB空间。所有的移动浏览器都会在征得用户同意后才允许网站使用缓存。桌面浏览器中，可能会有一个250 MB或350 MB的起始配额。如果缓存使用量增大，一些桌面浏览器会略微提高一点配额，但不保证会这样。

11.2 实用缓存技术

到现在为止，我们已经介绍了把一组页面和资源打包成离线应用的方法。期间，我们学习了如何编写和更新描述文件，以及如何让浏览器不要忽视我们的劳动成果。利用这些知识，很容易做出一个简单的离线应用来。可是，要实现复杂一些的站点的离线功能，仅有这些知识还不够。比如，我们想让某些内容在线，而在离线时将它们替换成其他页面，这就涉及如何（在代码中）判断计算机是否处于联网状态。在接下来的几节中，我们就来学习怎样编写更智能的描述文件，怎样通过简单的JavaScript检测设备在线状态。

11.2.1 访问未缓存的文件

经过前面的学习，我们知道浏览器在缓存了某个页面后，它就不会再向Web服务器发送请求，而是直接使用缓存的页面。但你知道吗，浏览器对离线页面的所有资源也持同样的态度，无论它是否缓存了这些资源。

比如，假设有个页面使用了两张图片，标记如下：

```
<img src="Images/logo.png" alt="Personality quiz">
<img src="Images/emotional_bear.jpg" alt="Sad stuffed bear">
```

可是，描述文件只要求浏览器缓存了一张图片：

```
CACHE MANIFEST
PersonalityTest.html
PersonalityTest_Score.html

PersonalityTest.css
PersonalityTest.js

Images/emotional_bear.jpg
```

有读者认为，浏览器会从缓存中取得emotional_bear.png，然后（在计算机联网的情况下）从Web服务器上取得logo.png。毕竟，过去的经验告诉我们，在从缓存的页面中访问未缓存的页面时，浏览器就会这样做。可是对于离线应用来说，没有这回事儿。事实上，无论什么浏览器，都会从缓存中取得emotional_bear.png，而忽略未缓存的logo.png并显示未找到文件的图标或者一块空白区域，具体显示什么，取决于浏览器。

要想解决这个问题，必须在描述文件中添加一个区块。这个区块的开头冠以“NETWORK:”字样，然后紧跟着一组必须在线访问的页面：

```
CACHE MANIFEST
PersonalityTest.html
PersonalityTest_Score.html

PersonalityTest.css
PersonalityTest.js
Images/emotional_bear.jpg

NETWORK:
Images/logo.png
```

这样，在联网时，浏览器才会尝试从Web服务器下载logo.png文件，而在离线时，则会忽略它。

此时，你可能会想：为什么要把不想缓存的文件都给列出来呢？或许是因为缓存空间有限的原因，比如，为了确保应用可以被那些只有少量缓存空间的浏览器缓存，你可能会考虑不让浏览器缓存那些大文件（参见11.1.5节）。

但更有可能是这些内容不能缓存，比如跟踪脚本或动态生成的广告。此时，最简单的办法是在“NETWORK:”区块中使用一个通配符，即星号（*）。这样浏览器就知道所有未缓存的内容都必须联网访问：

```
NETWORK:
*
```

另外，还可以使用星号匹配任意类型的文件（比如，*.jpg匹配所有JPEG图片），或者位于特定服务器上的所有文件（比如，http://www.google-analytics.com/*匹配Google Analytics域中的所有文件）。

注意 既然可以使用通配符，那在缓存文件列表中使用它可以吗？这样不必逐个罗列，就可以缓存一大批文件了。很遗憾，缓存文件列表不支持通配符，因为HTML5规范制定者担心有人会无意中缓存庞大的站点。

11.2.2 添加后备内容

利用描述文件可以告诉浏览器哪些文件要缓存，并通过NETWORK区块告诉哪些文件总是要从Web服务器获取且永不缓存。除此之外，描述文件还支持一个“FALLBACK:”区块，这里列出的文件可以根据计算机是否在线而互换。

“FALLBACK:”区块可以在描述文件中的任何地方出现，但要每行列出一对文件来。第一个文件名是在线时使用的文件名，第二个文件名是离线后备文件名。

```
FALLBACK:
PersonalityScore.html PersonalityScore_offline.html
```

浏览器会把后备文件（即这里的PersonalityScore_offline.html）下载并缓存起来。不过，只有在不能上网的时候浏览器才会使用这个后备文件。而在能上网的时候，浏览器会照常向Web服务器请求另一个文件（即这里的PersonalityScore.html）。

注意 不必为了让Web应用觉得“离线”而断开网络连接。实际上，关键在于能否访问到服务器，如果服务器没有响应，Web应用就会认为已经离线了。

至于什么时候该使用后备内容，那可能性就多了去了。比如，可以在离线时让浏览器使用一个简单点的页面，其中的脚本与在线页面中的不同，或者使用了更少的资源。后备内容放在哪儿都可以，只要一开始加上“FALLBACK:”就可以：

```
CACHE MANIFEST
PersonalityTest.html
PersonalityTest_Score.html

PersonalityTest.css

FALLBACK:
PersonalityScore.html PersonalityScore_offline.html
Images/emotional_bear.jpg Images/emotional_bear_small.jpg
PersonalityTest.js PersonalityTest_offline.js

NETWORK:
*
```

注意 在描述文件中，我们想要缓存的文件位于CACHE区块中。不过，除非你想要在另一个区域之后再列出要缓存的文件，否则不必有意添加这个区块。

后备内容区块也支持通配符匹配。这样就可以创建一个内置的错误页面，比如：

```
FALLBACK:
/ offline.html
```

这一行告诉浏览器对任何不在缓存里的文件使用后备文件。

假设有人要请求与离线应用在同一个网站中的页面，但该页面没有在缓存中。如果计算机在线，浏览器就会联系服务器取得该页面。如果计算机不在线，或者无法访问网站，或者根本就没有找到请求的页面，那浏览器会显示缓存的offline.html页面（图11-4）。



图11-4：浏览器访问的ImaginaryPage.html不存在，但浏览器却没有更新地址栏，因此用户无从知晓错误页面到底叫什么名字

在前面的例子中，使用一个斜杠（/）表示任意网页可能会让人觉得有点不太对劲儿，因为“NETWORK:”区块的通配符是星号。（不过，在有的浏览器中，比如Firefox，确实可以在后备区块中使用星号。）就当这是一个无害的HTML5怪癖吧。

顺便说一下，还可以通过用斜杠指定子目录来匹配更小范围内的文件：

```
FALLBACK:
/paint_app/ offline.html
```

在线时如何绕开缓存

加载一个已缓存的页面时，浏览器会期望从缓存中找到一切，而不管当时是在线还是离线。浏览器倾向于使用缓存并期望用它找到一切，除了那些明确写在NETWORK:区块的文件。

这种方式简单直接，但也不够灵活。如果你更希望在线时从线上加载页面，但离线时使用缓存文件，那么主要问题就来了。比如，设想一下一个新闻网站的前端页面，在线时，每次访问都从线上加载页面的最新版很合理；但如果离线了，最新的缓存页面可能还用得上。但标准的缓存机制不允许这种场景，因为它强制你在**永远**缓存和**绝不**缓存之间做一个抉择。

在HTML5标准化进程的最后一刻，这个解决方案出现了，添加一个叫SETTINGS:的区块，写法如下：

```
SETTINGS:
prefer-online
```

这告诉浏览器尽可能从线上获取资源，如果请求失败则使用缓存文件。

虽然这个权宜之计看上去还不错，但它有一些问题。这是一个应用于**每个**页面和资源（不仅仅是所选的文件）的全或无的设置。这使得在离线模式下使用缓存没那么快了，因为浏览器至少要浪费点时间尝试连接Web服务器。但最大的问题是，在写作本书的时候，只有Firefox支持prefer-online设置，其他浏览器会忽略它并依然用普通的方式使用缓存。

11.2.3 检测连接

孰不知，使用JavaScript检测浏览器当前是否在线的一个诀窍，就是利用后备区块。如果你是一位JavaScript老手，可能知道navigator.onLine属性，这个属性能够告诉你浏览器当前是否在线，

但不一定准确。onLine属性的问题是，它只真实地反映浏览器“脱机工作”的设置，并不反映计算机是否真的连到了因特网。就算onLine属性能真实反映连接情况，它也不会告诉你浏览器到底是没有连接到Web服务器，还是由于种种原因没有下载到网页。

所以，我们只能利用后备区块，让浏览器根据应用是否在线分别加载相同JavaScript函数的不同版本。为此，要在后备区块中添加如下文件对：

```
FALLBACK:
online.js offline.js
```

原始的网页引用online.js：

```
<!DOCTYPE html>
<html lang="en" manifest="personality.manifest">
<head>
  <meta charset="utf-8">
  <title>...</title>
  <script src="online.js"></script>
  ...
```

这个JavaScript文件包含着一个非常简单的函数：

```
function isSiteOnline() {
  return true;
}
```

如果浏览器没有下载到online.js，就会使用offline.js，后者包含着一个同名函数，但返回值不同：

```
function isSiteOnline() {
  return false;
}
```

在原始的网页中，为了知道应用是否在线，检测这个isSiteOnline()函数即可：

```
var displayStatus = document.getElementById("displayStatus");
if (isSiteOnline()) {
  //（可以运行依赖上网的任务，比如通过XMLHttpRequest连接Web服务器）
  displayStatus.innerHTML = "You are connected and the web server is online.";
}
else {
  //（应用在离线运行，需要隐藏或修改一些内容，或者禁用某些功能）
  displayStatus.innerHTML = "You are running this application offline.";
}
```

11.2.4　用JavaScript监听更新

使用相对有限的JavaScript接口可以与离线应用功能交互。这个JavaScript接口就由applicationCache对象定义。

通过applicationCache对象的status属性，可以知道浏览器当前在干什么，是在检测更新的描述文件，还是在下载新文件，抑或在做其他事。这个属性变化很快，也很有用；同样有用的是与不同属性值对应的事件（参见表11-1），这些事件会在applicationCache的状态变化时触发。

表11-1　缓存事件

事　件	说　明
onChecking	浏览器在发现网页中的manifest属性时，会触发这个事件，并向Web服务器请求描述文件
onNoUpdate	如果浏览器已经下载了描述文件，而描述文件并未改变，浏览器就会触发这个事件，然后不再做什么了
onDownloading	浏览器在开始下载描述文件（以及其中列出的文件）之前会触发这个事件。除了第一次下载描述文件时，更新文件时都会触发这个事件
onProgress	下载文件期间，浏览器会不断地触发这个事件，以报告进度
onCached	当新离线应用的所有文件第一次下载完成后，会触发这个事件。此后，不会再发生事件
onUpdateReady	这个事件表示已经取得了更新的内容。此时，新内容已经可以使用了，但除非重新加载页面，否则不会在浏览器窗口中出现。此后，不会再发生事件
onError	缓存期间发生任何问题都会触发这个事件。可能是无法连接Web服务器（这种情况下，会把页面切换到离线浏览模式），或者描述文件包含错误的语法，或者缓存的资源不存在。此后，不会再发生事件
onObsolete	在检测更新时，浏览器发现描述文件不存在了，就会触发这个事件。然后，它会清除缓存。下次再加载页面时，浏览器会从Web服务器取得实时、最新的在线版页面

图11-5展示了请求一个缓存页面时这些事件如何展开。

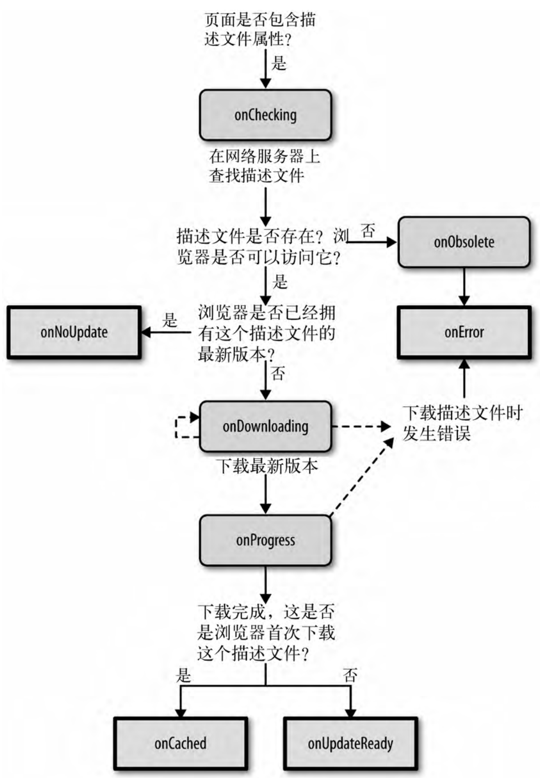


图11-5：如果你内心是个微观管理控，可以控制一下缓存过程。但首先你需要理解缓存事件是如何展开的，就像这幅图所描述的那样。这个流程会在浏览器到其中一个（粗边框的）方盒子时结束

这里面最有用的事件是onUpdateReady，表示浏览器已经下载了新版本的应用。即使新版本已经可以使用了，但浏览器窗口中显示的仍然是旧版本的内容。此时，可以利用这个事件告诉访客刷新页面，浏览新版本的内容，就像桌面应用下载完更新之后所做的那样：

```
<script>
window.onload = function() {

  //给onUpdateReady事件注册事件处理程序
  applicationCache.onupdateready = function() {
    var displayStatus = document.getElementById("displayStatus");
    displayStatus.innerHTML = "There is a new version of this application. " +
      "To load it, refresh the page.";
  }
}
</script>
```

要不，也可以使用window.location.reload()方法，在用户确认后重新加载页面：

```
<script>
window.onload = function() {

  applicationCache.onupdateready = function() {
  if (confirm(
   "A new version of this application is available. Reload now?")) {
      window.location.reload();
    }
  }
}
</script>
```

图11-6展示了这段代码的运行结果。

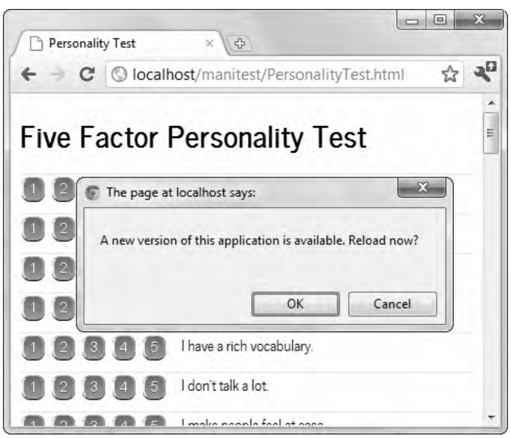


图11-6：如果访客单击OK，应用就会重新加载当前页面，显示更新后的内容（否则，下次再打开这个页面或者刷新页面之后，才会出现新内容）

除了status属性和上述事件之外，applicationCache对象还有两个方法：update()和

swapCache()。其中，update()方法的名字有点含糊，实际上调用它只会让浏览器检测是否有新的描述文件。如果有，浏览器就会在后台下载新文件；否则，什么也不做。

虽然浏览器能自动检测更新，但你也可以调用update()方法让它去检测，以便及时发现更新的描述文件。这个方法很适合那些生命期长的Web应用，比如一打开就是一整天的页面。

第二个方法是swapCache()，用于告诉浏览器开始使用新缓存的内容——如果它已经下载完了更新。然而，swapCache()方法不会影响当前显示的页面；要让当前页面显示新内容，必须重新加载它。那swapCache()还有什么用呢？通过切换到新缓存，此后加载的所有内容（比如动态加载的图片），都会从新缓存（而不是旧缓存）中取得。如果处理得好，利用swapCache()既可以让页面访问新内容，又不必强制完全重载（同时也就不会把当前应用重置为初始状态）。但在大多数应用中，使用swapCache()还是弊大于利，有时候会造成混用新、旧缓存的问题。

第 12 章

与Web服务器通信

本书一开始先介绍了HTML5中与标记有关内容，如语义元素、Web表单和视频。但随着学习的深入，我们逐渐接触了网页编程技术，以及HTML5中与JavaScript密切相关的知识。这一章，我们进一步探讨网页编程技术，不仅会涉及JavaScript代码，而且会有服务器端编程代码（即编写在Web服务器上运行的程序），但与服务器端编程语言无关。

讨论服务器端编程有一个问题。一方面，选择什么语言并不重要，只要它能操作纯HTML5页面即可（所有服务器端语言都可以）。另一方面，深入讲解一门你并没打算使用的或者你的Web主机根本不支持的语言，也没有什么必要。况且，全面讲解PHP、ASP.NET、Ruby、Java、Python等服务器编程的好书多得是。

本章讨论的问题需要的服务器端代码并不多，也不难懂。我们的设想是让这些代码刚好够演示每个HTML5功能，能够与网页中的JavaScript代码配合。对于你自己开发的项目，需要修改并扩展本章的示例，从而满足你自己的需要，或者适应你自己喜爱的语言。

说了半天，到底是什么功能需要用到服务器端编程呢？HTML5为网页与服务器通信提供了两种方式。第一种方式是服务器发送事件（Server-Sent events），让Web服务器能够定时给网页发送消息。第二种方式是Web Socket框架，让浏览器与Web服务器能够随心所欲地双向通信。不过，在讨论HTML5的这两项新功能之前，我们要先讲一讲当前广泛使用的服务器通信机制，即`XMLHttpRequest`对象。

12

注意　服务器发送的事件与Web Socket可不像看起来那么简单。要学会使用和编写简单的例子（本章的例子都是简单的例子）不难。但要利用它们构建一些专业网站中的功能，还要安全可靠，那可就是另外一码事儿了。关键在于，要在网站中实现这些功能，必须得有足够丰富的服务器端编程经验。

12.1　向 Web 服务器发送消息

在学习HTML5提供的与服务器通信的新功能之前，必须先了解此前与服务器通信的技术。当然，我们想要说的就是`XMLHttpRequest`这个不可或缺的JavaScript对象，页面利用它与Web服务

器通信。如果读者了解XMLHttpRequest对象（而且也在使用它），可以跳过这部分内容。不过，假如你一直都在从事静态网页设计，最好是读一读本节。

与Web服务器通信的历史

Web 诞生初期，与服务器通信是一件很简单、很平常的事儿。浏览器请求网页，然后服务器发回响应。仅此而已。

后来，软件公司的一些人想出了新点子。他们设计了服务器端工具，在第一步（请求网页）和第二步（返回页面）之间插入一些代码。这样做是为了动态地修改页面，比如在页面中间插入一段标记，甚至是生成一个新页面，比如从数据库中读取数据，然后生成特定的产品详细信息 HTML 页。

可是，Web 开发人员的目标更高，希望构建交互性更强的页面。服务器端编程也可以做到这一点，但实现起来有点费劲，而且浏览器必须不断刷新页面。比如，向购物车中添加一件商品，单击提交按钮可以提交当前页面（使用第 4 章介绍的表单），请求一个新页面。Web 服务器可以发回同一个页面，也可以发回一个不同的页面（比如，显示购物车中商品的新页面）。这样当然也很好，可就是有点烦琐。

Web 开发人员的要求又提高了，他们想实现更流畅的 Web 应用（比如电子邮件程序），而不是反复发送页面并且从头生成所有内容。实现这个想法的一组技术叫 Ajax，其中涉及一个叫 XMLHttpRequest 的 JavaScript 对象。使用这个对象，网页可以直接与 Web 服务器通信，发送数据，取得响应，不必整体提交或刷新。这样就让 JavaScript 真正具备了处理网页的能力，包括更新页面内容。而且，这种网页看起来会更流畅，反应也更快。

12.1.1　XMLHttpRequest对象

实现网页与Web服务器直接对话的关键是XMLHttpRequest对象。这个对象最早是由微软创造的，当时是想通过它改进Outlook电子邮件程序的Web版。但后来，所有现代浏览器都实现了这个对象。今天，XMLHttpRequest对象已然成为大多数现代Web应用的基础。

XMLHttpRequest对象的基本思想是让JavaScript代码自己发送请求，以便随时获取数据。这种请求是异步的，也就是说请求期间网页仍然能响应用户操作。事实上，网页用户根本不会注意到后台发送了请求（除非页面上会显示某些消息或进度条）。

XMLHttpRequest对象非常适合从Web服务器取得某些数据，下面是一些例子。

- **保存在服务器上的数据**。包括文件中的数据或数据库中的数据（如产品或客户记录）。
- **只有服务器才能完成计算的数据**。服务器可以执行复杂代码，完成复杂计算。虽然在客户端用JavaScript也能执行相同计算，但却不一定合适。有时候是因为JavaScript的数据计算功能没有那么强大，或者客户端不容易得到要拿来计算的数据。有时候是因为数据非常机密，必须防止别人偷看或者篡改。还有时候是因为计算量极大，客户端计算机不可能像高配置的Web服务器处理速度那么快。在这些情况下，让Web服务器完成计算是最合适的。

- **其他人Web服务器上的数据**。你的网页不能直接访问别人Web服务器上的数据。但是，你可以（通过XMLHttpRequest）调用自己Web服务器上的程序，让它帮你从其他服务器取得数据，然后再交给你。

要想真正理解XMLHttpRequest，最好的办法就是实际地使用它。接下来几节会展示两个简单的例子。

12.1.2 向Web服务器提问

图12-1展示了一个页面，可以让服务器做简单的算术题。这个表单的数据是通过XMLHttpRequest对象发送的。



图12-1：单击Ask the Server按钮，网页就会创建一个XMLHttpRequest对象，把两个数发送到服务器。服务器上有一个简单的脚本，计算出结果后会返回结果（见图12-2）

在创建这个页面之前，必须先写一段服务器端脚本，用来处理发送过去的数据（也就是用户输入的两个数值），然后返回结果。这个简单的任务是有史以来的任何一门服务器端编程语言都可以胜任的（发送一点点文本内容，总比发送一个完整的HTML文档要简单多了）。我们这个例子用的是PHP脚本，主要因为PHP相对简单，几乎所有网站托管公司都支持它。

1. 创建脚本

创建PHP脚本的第一件事儿是创建一个文本文件。在这个文件里，先写出如下所示的一个好玩的代码块，这样就确定了脚本的起始和结束位置：

```
<?php
  //这里是脚本代码
?>
```

我们这个例子的脚本代码很简单，就这么几行代码：

```
<?php
  $num1 = $_GET['number1'];
  $num2 = $_GET['number2'];
  $sum = $num1 + $num2
  echo($sum);
?>
```

就算你不是一个PHP专家，一看也该知道这几行代码能做什么。一开始当然是取得网页发送过来的两个数值：

```
$num1 = $_GET['number1'];
$num2 = $_GET['number2'];
```

美元？只是一个$符号而已。这里的$可不表示钱，在PHP中它表示变量名。换句话说，这两行代码声明了两个变量：`$num1`和`$num2`。为了给它们赋值，代码从内置的集合`$_GET`中取得数据。`$_GET`集合中保存着请求这个脚本的URL中包含的所有信息。

还是看个例子吧。如果你把这个PHP脚本放在一个名为WebCalculator.php的文件中，然后向它发送了如下请求：

```
http://www.magicXMLHttpRequest.com/WebCalculator.php?number1=34&number2=46
```

在这里，这个URL的末尾附加了两个信息，就在URL中叫做*查询字符串*的那一段（在代码里加点空格更方便解释，如下所示。现实情况下，浏览器不允许URL含空格）。

```
http://www.magicXMLHttpRequest.com/WebCalculator.php ? number1=34 & number2=46
```

首先，查询字符串中有一个名为`number1`的值，该值为34；接下来是名为`number2`的值，该值为46。这两个值前面的问号（?）表示查询字符串的开头，查询字符串中的和号（&）用于分隔每个名值对（除非查询字符串中只有一个值，那就不需要了）。PHP引擎接收到请求后，会把查询字符串中的值保存到`$_GET`集合中，以方便脚本代码访问。（大多数服务器端脚本语言都支持类似的数据模型。比如，微软的ASP.NET会把相同的信息保存在`Request.QueryString`集合中。）

注意 *HTML老手知道向Web服务器发送数据有两种方式，一种是像这样通过查询字符串，另一种是把数据放在请求的消息体中。无论采取哪种方式，数据编码不变，而且在服务器端访问这些数据的差别也不大。具体来说，访问消息体中的数据，在服务器端要访问`$_POST`集合，而不是`$_GET`集合。*

PHP脚本得到这两个数值后，会把它们加起来并且把结果保存到一个新变量中。这里的新变量是`$sum`：

```
$sum = $num1 + $num2
```

最后一步就把计算结果返回发送请求的网页。可以把结果打包放在一段HTML标记中，甚至可以格式化为XML。不过我们这个例子不必搞这么复杂，因为纯文本就足够了。但无论是有格式的，还是没有格式的，返回数据都一样要调用PHP的echo命令：

```
echo($sum);
```

就这么多，总共才4行PHP代码。可是，就凭这4行代码已经足以支撑B/S模式了：浏览器发送请求，Web服务器返回结果。

注意 能否完全用JavaScript来写这个例子，让它在浏览器运行，而不向Web服务器发送请求呢？当然可以。不过，这个例子的重点并不在于展示简单计算，关键是我们想通过它来说明服务器也可以完成计算任务。而且，无论PHP脚本写得多复杂，浏览器与服务器交换数据的基本模式永远不会变。

2. 请求服务器

第二步就是构建使用刚才PHP脚本的页面，这里要用到XMLHttpRequest对象。其实非常简单。首先，我们要创建一个XMLHttpRequest对象，以便在所有函数中访问它：

```
var req = new XMLHttpRequest();
```

在用户单击Ask the Server按钮时，会调用askServer()函数：

```
<div>
  <p>Enter two numbers:
    <input id="number1" type="number">
    <input id="number2" type="number">
  </p>
  <p>To ask the server to add them, without refreshing the page, click
  this button:<button onclick="askServer()">Ask the Server</button>
  </p>
</div>
<p id="result"></p>
```

这才是真正要做的事，这个askServer()函数会使用XMLHttpRequest对象在后台发送请求。首先，这个函数会收集所需的数据——两个数值，它们在表单的文本输入框中：

```
function askServer() {
  var number1 = document.getElementById("number1").value;
  var number2 = document.getElementById("number2").value;
```

然后，使用这两个数值构建一个查询字符串，用前面介绍的那种格式：

```
var dataToSend = "?number1=" + number1 + "&number2=" + number2;
```

接下来该准备发送请求了。调用XMLHttpRequest对象的open()方法是第一步。这个方法接收三个参数：HTTP操作类型（GET或POST）、作为请求目标的URL和表示浏览器是否异步工作的布尔值（true或false）：

```
req.open("GET", "WebCalculator.php" + dataToSend, true);
```

注意 说到第三个参数，懂行的读者可能会不禁担心起来："open()方法的最后一个参数应该只传入true啊，就要是打开异步工作模式。"没错，因为我们不能保证接收请求的网站完全没有问题，如果打开同步工作模式（即强制代码停止执行，一直等到服务器响应）可能会因为等不到响应而导致网页崩溃。

在真正发送请求之前，还必须为XMLHttpRequest对象的onReadyStateChange事件指定回调函数。只要服务器返回信息，就会触发这个事件，其中包括返回响应数据。下面这段代码将回调函

数设置为页面其他地方定义的一个函数，名为handleServerResponse():

```
req.onreadystatechange = handleServerResponse;
```

然后，就可以调用XMLHttpRequest对象的send()方法实际地发送请求了。因为是异步请求，所以后面的代码会紧跟着执行，不会停顿。而我们获得响应的唯一途径就是通过onReadyStateChange事件，而该事件稍后才会触发：

```
  req.send();

  document.getElementById("result").innerHTML = "The request has been sent.";
}
```

当onReadyStateChange事件触发并且接收到响应时，需要首先检测XMLHttpRequest的两个属性。一个是readyState，它的值是从0到4的一个数值，分别表示请求已经初始化完毕（1）、请求已经发送（2）、已经接收到部分响应（3）和请求响应完成（4）。显然，除非readyState属性的值为4，否则往下处理没有意义。另一个属性是status，保存着HTTP状态码。对这个属性，我们需要等待它的值变成200，表示一切顺利。假设我们请求的是一个网页，那么HTTP状态码可能是401（表示不允许访问）、404（没有找到）、302（已经移动）或503（服务器繁忙），等等。（要了解完整的HTTP状态码，请参考：www.addedbytes.com/for-beginners/http-status-codes。）

我们的例子是这样检测这两个属性的：

```
function handleServerResponse() {
  if ((req.readyState == 4) && (req.status == 200)) {
```

如果两个条件都满足，就可以从XMLHttpRequest的response属性中取得结果了。当然，我们知道，这个结果是两个数值相加的和。然后，代码会把这个结果显示在页面上（参见图12-2）：

```
    var result = req.responseText;
    document.getElementById("result").innerHTML = "The answer is: " +
     result + ".";
  }
}
```

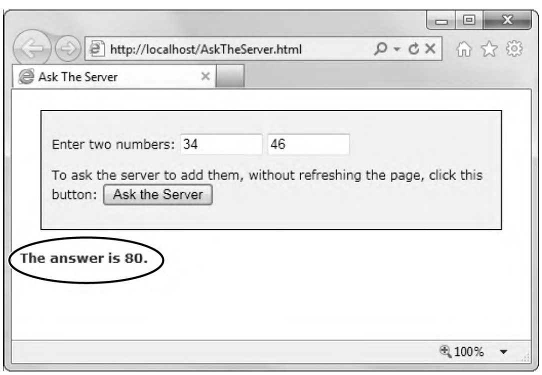


图12-2：Web服务器返回响应触发了JavaScript函数，函数把结果显示在网页上

XMLHttpRequest对象其实并不限制你请求的数据类型，其名字中带XML是因一开始设计它的

时候是要处理XML数据，因为XML是组织结构化数据的一种方便、有语义的格式。可是，XMLHttpRequest也可以请求简单的文本（就像我们这个例子中这样）、JSON数据（参见10.2.4节）、HTML（下一个例子会涉及）和XML等。事实上，目前用XMLHttpRequest请求非XML数据的情况反而更多见，因此提醒大家不要被它的名字所迷惑。

提示 在调用服务器端代码之前，必须先把包含PHP脚本的网页放到一个测试服务器上。为了简单起见，欢迎读者到本书网站上试验：http://prosetech.com/html5。

12.1.3 取得新内容

XMLHttpRequest对象的另一种用法是取得新的HTML内容，然后插入到当前页面中。比如，新闻报道中可能包含多张图片，但每次只显示其中一张。按一下按钮，JavaScript就会取得下一张图片，然后插入到页面中。再比如，页面中有一个“五强”或“十强”的幻灯片，每次单击链接都会显示一张。图12-3展示了一个与文章相配的幻灯片，每张图片都配有相应的文字说明。

图12-3所示的用例在很多情况下都适用。在恰当设计的情况下，这种方法非常适合展示大量内容，既能保证可读性，又不至于让人觉得内容过多。（如果内容本身不多，这样做只会徒增复杂性，迫使用户发送多个请求才能得到完整内容。）

取得部分内容最好是使用XMLHttpRequest对象。因为使用XMLHttpRequest取得部分内容后再更新页面，不会引起整个页面的刷新。就为了更新局部内容而刷新整个页面可不好，那样就会把其他没有更新的内容再重新下载一遍，导致页面闪烁，而且有时候还会把页面重新定位到顶部。这些说起来似乎是小事，没那么严重，但正是这些细节决定了人们对你网站的印象是流畅快捷，还是老气横秋、不可救药。

12

要做出图12-3所示的例子，首先要给动态内容留出空间来。下面这个<div>元素背景是金黄色的，它下面是两个链接：

```
<div id="slide">Click Next to start the show.</div>
<a onclick="return previousSlide()" href="#">&lt; Previous</a> 
<a onclick="return nextSlide()" href="#">Next &gt;</a>
```

这两个链接分别调用previousSlide()和nextSlide()函数，表示向前或向后导航。这两个函数的用途是增加计数器，计数器的值从0开始，最大为5，然后再递减回1。其中，nextSlide()函数的代码如下：

```
var slideNumber = 0;

function nextSlide() {
  //向前移动幻灯片索引
  if (slideNumber == 5) {
    slideNumber = 1;
  } else {
    slideNumber += 1;
  }
```

```
    //调用另一个函数显示当前幻灯片
    goToNewSlide();

    //取消链接的默认动作
    //（即不会打开新页面）
    return false;
  }
```



图12-3：这个页面把内容分成多个幻灯片。单击Previous或Next链接可以加载新幻灯片，包含新图片和相应的说明文本。实现这个功能的时候要用到XMLHttpRequest对象，它会在必要时请求新内容

而previousSlide()函数的代码与之非常类似：

```
function previousSlide() {
  if (slideNumber == 1) {
    slideNumber = 5;
  } else {
    slideNumber -= 1;
  }

  goToNewSlide();
  return false;
}
```

这两个函数都依赖于另一个函数goToNewSlide()去完成真正的工作。所谓真正的工作就是它会使用XMLHttpRequest与Web服务器通信，然后取得新数据。

一个现实的问题：ChinaSites.html页面从哪里取得数据？复杂一点的话，它可以从某个Web服务或PHP脚本取得数据。新内容可以动态生成，或者从数据库中读出来。但我们这个例子会采取技术含量低一点的方法，但在任何Web服务器上都行得通——读取特定的文件。比如，与第一张幻灯片对应的文件是ChinaSites1_slide.html，与第二个张幻灯片对应的文件是ChinaSites2_slide.html，以此类推。每个文件中里包含一小段HTML标记（而不是一个完整的HTML页面）。例如，ChinaSites5_slide.html中包含如下HTML标记：

```
<figure>
  <h2>Wishing Tree</h2>
  <figcaption>Make a wish and toss a red ribbon up into the branches
  of this tree. If it sticks, good fortune may await.</figcaption>
  <img src="wishing_tree.jpg">
</figure>
```

既然知道了数据保存在哪里，通过XMLHttpRequest取得正确的文件就是小菜一碟了。为生成正确的文件名，只要在一行简单的代码中嵌入计数器当前的值即可。下面就是goToNewSlide()函数的代码：

```
var req = new XMLHttpRequest();

function goToNewSlide() {
  if (req != null) {
    //准备请求包含幻灯片数据的文件
    req.open("GET", "ChinaSites" + slideNumber + "_slide" + ".html", true);
    //设置用于处理幻灯片数据的函数
    req.onreadystatechange = newSlideReceived;

    //发送请求
    req.send();
  }
}
```

最后一步就是把取得的数据复制到表示当前幻灯片的<div>元素中：

```
function newSlideReceived() {
  if ((req.readyState == 4) && (req.status == 200)) {
```

```
    document.getElementById("slide").innerHTML = req.responseText;
  }
}
```

提示 为了让幻灯片更生动，可以创建过渡效果。比如，新图片可以淡入到视图，与此同时淡出旧图片。为此，只要修改opacity属性，用JavaScript计时器（参见9.4.1节）或CSS3过渡（参见6.3.3节）。（这里是经典的实现方案：http://clagnut.com/sandbox/imagefades.php，这里的CSS方案使用了新的CSS3过渡功能，但该功能尚未得到所有浏览器支持：http://css3.bradshawenterprises. com/cfimg2/。）实际上，这也是使用XMLHttpRequest的动态页面的一个好处——可以控制新内容呈现的方式。

这个例子到现在还没有完。13.3.3节在讨论HTML5的历史管理功能时，还会讲到如何让这个例子中当前显示的幻灯片与浏览器地址栏中的URL匹配。不过，就本章内容来说，我们接下来应该介绍其他与Web服务器通信的方式了。

注意 如果用目前流行的jQuery JavaScript工具包，可能不需要直接用XMLHttpRequest对象，而是用jQuery方法，比如jQuery.ajax()，它会在后台使用XMLHttpRequest，技术是一样的，但jQuery把过程简单化了。

12.2 服务器发送事件

使用XMLHttpRequest可以向Web服务器发送请求，并且能很快得到响应。这种通信方式是一对一的，即Web服务器响应之后，通信就结束了。换句话说，Web服务器不可能等几分钟，等有了更新之后再发送一次响应。

不过，有一些网页可以与Web服务器保持长期的联系。比如，显示股票报价的Google Finance（http://www.google.com/finance）。在桌面上打开这个页面不用管它，股票价格也会定期更新。再比如，英国广播公司（BBS）的滚动新闻页面http://www.bbc.co.uk/news/。呆在这个页面上一天，你也会发现新闻标题自动更新。当然，在一些Web邮件程序里，比如微软的Outlook.com（www.outlook.com）也能发现收件箱会不断增加新邮件。

以上例子中的网页使用了一种技术，叫做轮询。顾名思义，轮询就是每隔一定时间（比如几分钟）就向Web服务器请求新数据。为了实现这个操作，可以使用JavaScript的setInterval()或setTimeout()函数（参见9.4.1节），每过设定的时间就触发一次代码。

轮询是一个合理的方案，但有时候效率不高。因为轮询意味着要向服务器发送请求，要建立新连接，而这样做只是想知道是否有新数据。如果成千上万的用户都这样轮询，无疑会给服务器造成无谓的压力。

还有一种方案，叫做服务器发送事件，可以让网页与Web服务器保持连接。服务器任何时候都可以发送消息，而不必频繁断开连接，然后再重新连接并重新运行服务器端脚本。（除非你希望如此，因为服务器发送事件也支持轮询。）最关键的是，使用服务器发送事件很简单，大多数Web主机都支持，而且极其稳定可靠。但毕竟它是一个相对新的技术，在本书写作时Internet Explorer不支持它，如表12-1所示。

表12-1 浏览器对服务器发送事件的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android版Chrome
最低版本	—	6	6	5	11	4	29

注意 如果你想找一些能够模拟服务器发送事件的腻子脚本，可以参考这里：http://tinyurl.com/polyfills。

接下来几节，我们将通过一个简单的例子演示服务器发送事件。

12.2.1 消息格式

与使用XMLHttpRequest不同，服务器发送事件这个标准不允许随意发送数据，而是必须遵循一个简单但明确的格式。每条消息必须以data:开头，然后是实际的消息内容，再加上换行符（PHP等很多编程语言中用\n\n表示换行符）。

下面就是一条标准的通过因特网传输的消息：

```
data: The web server has sent you this message.\n\n
```

也可以把一条消息分成多行，每行都要跟一个行结束符，用\n表示。这样就可以发送较复杂的内容了：

```
data: The web server has sent you this message.\n
data: Hope you enjoy it.\n\n
```

不过，一条消息分成多行，每行开头仍要有data:，而整个消息结束同样要跟\n\n。

利用这个格式，甚至可以发送JSON数据（参见10.2.4节），这样网页只需一步即可把文本转换为JavaScript对象：

```
data: {\n
data: "messageType": "statusUpdate",\n
data: "messageData": "Work in Progress"\n
data: }\n\n
```

除了消息本身之外，Web服务器还可以发送唯一的ID值（使用 id: 前缀）和一个连接超时选项（使用retry:前缀）：

```
id: 495\n
retry: 15000\n
data: The web server has sent you this message.\n\n
```

你的网页一般只关注消息本身，不关心ID和连接超时信息。ID和超时信息是浏览器要使用的。比如，浏览器在读到以上信息后就会知道，如果连接已经断开，那么应该在15 000毫秒（15秒）后再重新建立连接。重新建立连接时，应该把ID编号495一起发给服务器，以便确认。

注意 浏览器与服务器失去联系的原因有很多，比如网络中断或者代理服务器等待数据超时。浏览器会在可能的情况下自动重新打开连接，默认等待时间为3秒钟。

12.2.2 通过服务器脚本发送消息

知道了消息的格式，编写服务器端脚本发送消息就不在话下了。同样，我们还是以几乎所有Web主机都支持的PHP来写一个直观的例子。图12-4展示了一个从服务器取得消息的页面，而消息的内容只是服务器上的当前时间。

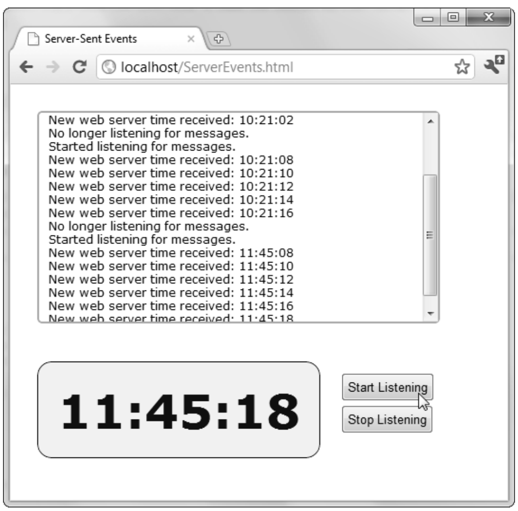


图12-4：页面在监听的情况下，会连续不断地收到服务器发送的消息，大约每两秒钟一条。每条消息都会在上面的消息框中依次列出，消息框下方的时间表示接收最后一条消息的时间

注意 Web服务器的时间是持续变化的，因此保存该时间信息的变量也要持续更新。为了演示这一点，我们就可以创建一个简单的服务器端事件。不过，在真正的开发中，服务器消息应该发送更有价值的消息，比如为生成滚动新闻而发送最新的新闻标题。

这个例子的服务器端代码只是简单地过两秒钟就返回一次时间，以下是完整的代码：

```
<?php
  header("Content-Type: text/event-stream");
  header('Cache-Control: no-cache');
  ob_end_clean();
```

```
  //开始不间断的循环
  do {
    //取得当前时间
    $currentTime = date("h:i:s", time());

    //把时间放到消息中发送
    echo "data: " . $currentTime . PHP_EOL;
    echo PHP_EOL;

    flush();

    //等两秒钟再创建新消息
    sleep(2);
  } while(true);
?>
```

脚本开始先设置了两个重要的头部信息。首先，设置了MIME类型为`text/event-stream`，这是服务器端事件标准规定的：

```
header('Content-Type: text/event-stream');
```

然后，告诉Web服务器（及代理服务器）关闭Web缓存。否则，含有时间的消息可能不会按先后次序到达：

```
header('Cache-Control: no-cache');
```

还需要一步来关闭PHP内置的缓冲机制，这样PHP脚本返回的数据会立即传给浏览器：

```
ob_end_clean();
```

剩下的代码构成了一个无限循环（至少在客户端存在的情况下会一直循环下去）。每次循环都会调用内置的`time()`函数，取得当前时间（格式为`hh:mm:ss`），并将其保存在一个变量中：

```
$currentTime = date('h:i:s', time());
```

接下来，循环利用这个变量按照正确的格式来构建一条消息，以便使用PHP的echo命令发送。这个例子中的消息只有一行，以`data:`开头，然后是时间。消息字符串的最后是一个常量`PHP_EOL`（在PHP中表示end of line，即行结束符），也就是我们前面讨论的`\n`字符：

```
echo 'data: ' . $currentTime . PHP_EOL;
echo PHP_EOL;
```

注意 或许有读者觉得用点操作符（.）来连接字符串很有意思。这里跟JavaScript中使用加号（+）操作符连接字符串是一样的；当然，使用加号连接字符串时，必须至少有一个操作数是字符串，如果每个操作数都是数值，那就会执行加法计算了。

调用`flush()`函数的用意是立即发送数据，而不是先缓冲起来，等到PHP代码执行完毕再发送。最后，`sleep()`函数会让程序暂停两秒钟，然后再继续下一次循环。

提示　如果接收两条消息会等待较长的时候，可能是连接被某个**代理服务器**（位于Web服务器与客户机之间，用于分散流量的服务器）给切断了。为了避免这种情况发生，可以每隔15秒左右，发送一条只包含冒号（:）而没有文本内容的注释消息。

12.2.3　在网页中处理消息

监听服务器发送消息的网页更简单。以下就是网页<body>部分的所有标记，分为三个<div>区块：一个用于滚动显示消息、一个用于显示时间，另一个用于显示按钮：

```
<div id="messageLog"></div>
<div id="timeDisplay"></div>
<div id="controls">
  <button onclick="startListening()">Start Listening</button><br>
  <button onclick="stopListening()">Stop Listening</button>
</div>
```

页面加载后，JavaScript代码会找到ID为messageLog和timeDisplay的<div>元素，将它们保存在全局变量中，以便后面的函数使用：

```
var messageLog;
var timeDisplay;

window.onload = function() {
  messageLog = document.getElementById("messageLog");
  timeDisplay = document.getElementById("timeDisplay");
};
```

监听事件的过程在用户单击Start Listening按钮的时候开始。此时，JavaScript会创建一个新EventSource对象，传入服务器端发送消息的脚本URL。（在这个例子中，脚本名为TimeEvents.php。）然后，将处理函数指定给onMessage事件，这个事件会在页面接收到消息时触发：

```
var source;

function startListening() {
  source = new EventSource("TimeEvents.php");
  source.onmessage = receiveMessage;
  messageLog.innerHTML += "<br>" + "Started listening for messages.";
}
```

提示　为了检测浏览器是否支持服务器端事件，可以测试是否存在window.EventSource属性。如果不存在，就要使用后备方案。比如，可以使用XMLHttpRequest对象定时向Web服务器请求数据。

在触发了receiveMessage函数时，可以从事件对象的data属性中取得消息。对我们这个例子而言，会把新消息显示在原来的消息框中，然后更新时间显示：

```
function receiveMessage(e) {
  messageLog.innerHTML += "<br>" + "New web server time received: " + e.data;
  timeDisplay.innerHTML = e.data;
}
```

注意，网页接收到的消息不会包含前缀data:和结束的\n\n符号，只有其中的消息内容（也就是时间值本身）。

最后，调用EventSource对象的close()方法，可以让页面停止监听服务器事件，相应的代码如下：

```
function stopListening() {
  source.close();
  messageLog.innerHTML += "<br>" + "No longer listening for messages.";
}
```

12.2.4 轮询服务器端事件

前面的例子以最简单的方式使用了服务器端事件：页面发送请求，连接保持打开，服务器定时发送信息。在当前连接有问题或者出于其他目的（比如手机电池快没电了）临时终止了通信时，浏览器可能需要重新连接（重新连接也是自动的）。

如果服务器脚本结束了，而且服务器关闭了连接怎么办？这个就有意思了，因为即便服务器有意关闭连接，网页仍然会自动重新打开连接（默认等待3秒钟），再次请求脚本，然后从头开始。

这种机制是可以利用的。比如，假设你写了一个比较短的服务器脚本，只发送一条消息。而此时网页就像在使用轮询（参见前几页），周期性地重新建立连接。唯一的差别就是Web服务器会告诉浏览器再等待多长时间才能检查新数据。在真正使用轮询的网页中，等待时间是在JavaScript代码中确定的。

下面这段脚本混合了两种手段，它保持连接（并周期性地发送消息）1分钟，并建议浏览器等待2分钟再重新连接，然后关闭连接：

12

```
<?php
  header("Content-Type: text/event-stream");
  header('Cache-Control: no-cache');

  ob_end_clean();

  //告诉浏览器在连接关闭后
  //等待2分钟再重新连接
  echo "retry: 120000" . PHP_EOL;

  //保存开始时间
  $startTime = time();

  do {
    //发送消息
    $currentTime = date("h:i:s", time());
    echo "data: " . $currentTime . PHP_EOL;
    echo PHP_EOL;
    flush();
```

```
    //如果过了1分钟，结束脚本
    if ((time() - $startTime) > 60) {
      die();
    }

    //等5秒钟，发送新消息
    sleep(5);
  } while(true);
?>
```

这样，等再运行脚本时，就可以做到用1分钟时间来更新，然后暂停服务2分钟（参见图12-5）。对于较复杂的例子而言，或许需要Web服务器向浏览器发送一条特殊的消息，告诉它不必等待数据更新（比如股市今天已经停止交易了）。此时，网页就可以调用EventSource对象的close()方法。



图12-5：这个页面使用了消息流（在1分钟时间内发送一批消息），然后使用轮询（等待2分钟）。这种方案有助于减少Web服务器的流量，但要考虑数据更新的频率及保持数据最新的必要程度

注意　对于复杂的服务器脚本，浏览器的自动重新连接行为有时候并不好对付。比如，Web服务器可能会在执行某个任务期间就把连接断开。这种情况下，Web服务器代码会给每个客户端发送一个ID（如12.2.2节所述），以便重新连接时再把这个ID发给服务器。可是，服务器端代码必须负责生成ID、记录每个ID的操作（比如把某些数据保存到数据库里），以及在停止处进行恢复等。所有这些都需要你具有丰富的编码经验。

12.3 Web Socket

服务器发送事件非常适合从服务器连续不断地接收消息。但整个通信完全是单向的，无法知道浏览器是否响应，也不能进行更复杂的对话。

如果你想创建一个应用，浏览器与服务器需要正式对话，那你很可能使用XMLHttpRequest对象（而不用Flash）。使用XMLHttpRequest对象在很多情况下没有问题，但同样也有很多情况不合适。首先，XMLHttpRequest不适合快速地来回发送多条消息（比如，聊天室）。其次，没有办法将一次调用与下一次调用联系起来，每次网页发送请求，服务器都要确定请求来自何方。在这种情况下，要想把一系列请求关联起来，服务器端代码会变得非常复杂。

有一个方案能解决这个问题，这不是瞎掰 。这个方案就是Web Socket标准。根据这个标准，浏览器能够保持对Web服务器打开的连接，从而与服务器长时间交换数据。Web Socket标准让开发人员非常兴奋，而且已经获得相当多的浏览器支持（见表12-2），但IE到IE10之后才支持。

表12-2　浏览器对Web Socket的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android版Chrome
最低版本	10	11	14	6	12.1	6	29

提示　Web Socket有点繁琐。比如，在支持它的浏览器上运行也会出问题，可能是因为电脑网络设置的限制、防火墙或者杀毒软件。如果不确定你的电脑是否能用Web Socket，用http://websocketstest.com测试一下。这个网站会尝试连接一个测试服务器，并且生成一个单页报告告诉你Web Socket是否在运行。

使用Web Socket之前，必须理解两点。第一，Web Socket是一种专用手段，非常适合开发聊天室、大型多人游戏，或者端到端的协作工具。利用它能开发出很多新应用，但恐怕不太适合今天JavaScript驱动的Web应用（比如电子商务站点）。

第二，Web Socket方案做起来可能会无比复杂。网页中的JavaScript很简单，可服务器端代码不好写，为此必须熟练掌握编程技能，而且要对多线程和网络模型有深刻理解，这些超出了本书的范围。但是，如果其他公司、服务商或者编程牛人已经创建了一个可用的Web Socket服务，你只需具备一点HTML5相关的JavaScript知识就能看懂代码了。接下来的几节会详细介绍。

12.3.1 Web Socket服务器

为了使用Web Socket，需要在Web服务器上运行一个程序（也叫Web Socket服务器）。这个程序负责协调各方通信，而且启动后就会不间断地运行下去。

> **注意** 很多Web主机不允许长时间运行程序，除非你购买的是**专用服务器**（只分配给你的网站使用，不与他人共享）。如果你使用的是共享主机，那很可能无法创建使用Web Socket的网页。就算你能启动Web Socket服务器，让它不间断运行，Web主机商会检测它并把它关掉。

为了让读者了解Web Socket服务器是干什么的，下面列出了与它有关的一些任务。

- 设置消息“词汇表”，即确定哪些消息有效，这些消息有什么含义。
- 记录当前连接的所有客户端。
- 检测向客户端发送消息是否出错，如果客户端已经停止响应，则终止与该客户通信。
- 处理内存数据，也就是所有Web客户端都可以访问的数据。可能涉及很多微妙的问题，比如一个客户端要加入，而另一个客户端要退出，这两个客户端的连接都保存在同一个内存对象中。

多数开发人员都没有利用Socket创建过服务器端程序，因为这样做明显代价太大。最简单的办法是安装别人写好的Web Socket服务器，然后再设计网页来与之通信。Web Socket标准的JavaScript部分使用起来很容易，不会有什么问题。而另一个部分可以采取别人写好的Socket服务器代码，根据需要稍加改动即可。目前，有很多现成的Web Socket服务器项目（其中不乏开源和免费的）。这些项目致力于实现各种功能，也支持很多种服务器端编程语言。12.3.3节将详细介绍一些项目。

12.3.2 简单的Web Socket客户端

从网页的角度看，Web Socket很容易理解，也很容易使用。第一步就是创建`WebSocket`对象，传入一个URL，比如：

```
var socket = new WebSocket("ws://localhost/socketServer.php");
```

这个URL以ws://开头，是一个表示Web Socket连接的新协议。不过，URL仍然指向服务器上的一个Web应用（即这里的socketServer.php脚本）。Web Socket标准还支持以wss://开头的URL，表示安全、加密的连接（这跟请求网页时使用https://而不是http://一样）。

> **注意** Web Socket并不仅限于连接自己的Web服务器。你的网页可以打开一个到其他Web服务器的Web Socket连接，而不会增加任何工作量。

创建`WebSocket`对象后，页面就会尝试连接服务器。下一步，就是使用`WebSocket`对象的4个事件：`onOpen`、`onError`、`onClose`和`onMessage`。其中，`onOpen`会在建立连接后触发，`onError`会在出现问题时触发，`onClose`会在连接关闭时触发，而`onMessage`会在页面从服务器接收到消息时触发。利用这些事件，就可以实现与Web Socket服务器的通信：

```
socket.onopen = connectionOpen;
socket.onmessage = messageReceived;
socket.onerror = errorOccurred;
socket.onopen = connectionClosed;
```

比如，连接成功后，需要向服务器发送一条消息。发送消息要使用WebSocket对象的send()方法，这个方法接收纯文本内容作为参数。下面就是处理onOpen事件并发送消息的函数：

```
function connectionOpen() {
  socket.send("UserName:jerryCradivo23@gmail.com");
}
```

可想而知，服务器在收到这条消息后，会发回一条新消息。

利用onError或onClose事件，可以向用户发出通知。不过，最重要的事件还是onMessage，每当Web服务器发来新数据时，都会触发这个事件。同样，响应这个事件的JavaScript代码也非常好理解，主要是从事件对象的data属性中取得消息内容：

```
function messageReceived(e) {
  alert("You received a message: " + e.data);
}
```

如果网页认为通信可以结束了，可以调用close()方法关闭连接：

```
socket.close();
```

但是，一旦socket关闭，就不能再发送消息了，除非重新创建WebSocket对象。重建socket对象和第一次创建一样——用new关键字，传入URL，并绑定所有的事件回调函数。如果需要频繁地连接与断开，可以把这些代码封装到独立的函数里以方便调用。

如你所见，WebSocket对象惊人地简单。事实上，我们已经介绍完它所有的方法和事件了。经过以上简单介绍，我们知道使用其他人写好的Web Socket服务器并不费事，只要知道发送什么消息，以及服务器会发回什么消息即可。

12

注意 建立Web Socket连接时，后台其实会执行很多处理工作。首先，网页要使用常见的HTTP标准与服务器建立联系，然后再把连接“升级”到Web Socket连接，以便浏览器与服务器能够双向通信。此时，如果计算机与服务器之间有代理服务器（比如在公司网络中），可能会遇到问题。代理服务器可能会拒绝请求并断开连接。对于这种问题，可以检测失败的连接（通过WebSocket对象的onError事件），然后使用http://tinyurl.com/polyfills中的Socket“腻子脚本”来作为后备。这些脚本会使用轮询来模拟Web Socket连接。

12.3.3 使用现成的Web Socket服务器

你想自己试试Web Socket？没问题，网上有很多Web Socket服务器可供你搭建试验环境。可以先试下www.websocket.org/echo.html，这个页面提供了一个最基本的Web Socket服务

器：你向它发送消息，它再返回同样的消息（参见图12-6）。虽然这谈不上有意思，但却能让我们熟悉WebSocket对象的所有功能。事实上，你还可以创建自己的页面和服务器交互，这是一种练习Web Socket编程技能的好方法。只要使用正确的Web Socket服务器URL（在这个例子中是ws://echo.websocket.org），代码就能正常运行，不管页面是在一个不同的域下还是存在本地电脑上。



图12-6：只会重复发送浏览器所发送消息的Web Socket服务器。虽然没有什么惊人的创意，但却能让你体验到与现成的Web Socket服务器对话有多么简单

下面这段JavaScript代码不难理解，第一个任务是在页面第一次加载完成时创建socket对象并且绑定所有的事件：

```
var socket;

window.onload = function () {
  connect();
}

function connect() {
  socket = new WebSocket("ws://echo.websocket.org")

  // 监听所有Web socket事件
  socket.onopen = connectionOpen;
  socket.onmessage = messageReceived;
  socket.onerror = errorOccurred;
```

```
  socket.onopen = connectionClosed;
}
```

这里，socket连接的代码移到了一个专门的函数connect()里，以便随时调用。这样就可以随心所欲地连接和断开。前面已经介绍过断开连接的代码：

```
function disconnect () {
  socket.close();
}
```

顺便说一下，可以查看socket对象的readyState属性来判断它是以下哪种状态：打开并且准备就绪（readyState是1）、还没建立（0）、关闭过程中（2）、已关闭（3）。

服务器没有真正要说的话，它对所有消息都做一样的处理，只是简单地把文本返回到页面。正因为如此简单，只需简单地在用户点击一个按钮时将文本输入框中当前的内容发送给服务器。

```
function sendMessage() {
  //获取消息数据
  var message = messageBox.value;

  //通过socket发送消息
  socket.send(message);

  //告诉用户刚刚发生了什么
  messageLog.innerHTML += "<br>" + "Sent: " + message;
}
```

将收到的消息插入页面也很容易：

```
function messageReceived(e) {
  messageLog.innerHTML += "<br>" + "Message received: " + e.data;
}
```

12

注意 如果想看更有趣一些的例子，可以访问http://html5demos.com/web-socket。登录这个简单的Web Socket服务器，发送一条消息，所有人都能立即收到消息。

Web Socket服务器

想要自己做一个实际的例子？那必须得有一个能与网页通信的 Web Socket 服务器。虽然如何编写 Web Socket 服务器超出了本书范围（一个服务器程序至少得写几十行代码），但网上有很多现成的测试服务器可以拿来用。下面就分别按语言列出几个来，供读者参考。

- **PHP**。这是一个简单的项目，在此基础上可以用 PHP 来编写一个 Web Socket 服务器。地址是：http://code.google.com/p/phpwebsocket。
- **Ruby**。针对 Ruby 的 Web Socket 服务器有不少，但这个使用了 EventMachine 模型，很受欢迎。地址是：http://github.com/igrigorik/em-websocket。
- **Python**。这是一个 Apache 扩展，使用 Python 增加了 Socket 服务器。地址是：http://code.google.com/p/pywebsocket。

- **.NET**。简单，可不能这么说。这个功能全面的项目包含了完整的 Web Socket 服务器实现，基于微软.NET 平台和 C#语言。下载地址是：http://superwebsocket.codeplex.com。
- **Java**。与刚才.NET 那个项目类似，但是用纯 Java 写的。地址是：http://jwebsocket.org/。
- **node.js**。不同的人可能会有不同的看法，node.js（能运行 JavaScript 的 Web 服务器）既是一个前途无量的新生事物，又是一个发展迅猛的测试工具。无论如何，我们可以找到一个能支持它的 Web Socket 服务器，地址是：http://github.com/miksago/node-websocket-server。
- **Kaazing**。与这里列出的其他项目不同，Kaazing 并不提供 Web Socket 服务器的代码，而是提供一个成熟的 Web Socket 服务器，经过许可你可以在自己的网站中使用它。对于想要冒险的开发人员，这个现成的工具可能不够刺激。但对于没有那么高要求的网站来说，使用它还是挺不错的，特别是它的客户端库还内置了对回调的支持（首先尝试使用 Web Socket 标准，然后再尝试 Flash，实在不行再使用 JavaScript 轮询）。详细信息，请参考这里：http://kaazing.com/products/html5-edition.html。

第 13 章

地理定位、Web Worker和历史管理

到目前为止，我们已经介绍了HTML5的所有关键主题。而你也已经使用它写出了更有意义、结构更清晰的标记。此外，本书还讨论了HTML5丰富的图形功能（视频和动态绘图），以及不能上网也照样可以使用的基于JavaScript的离线应用。

本章，我们再来介绍三个之前没有提到的JavaScript新功能。或许有读者了解过了，这些功能扩展了网页的能力，只要编写少量JavaScript代码即可。下面就请这三个新功能隆重登场。

- **Geolocation（地理定位）**。尽管经常以HTML5的名义提到，但地理定位实际上是一个单独的标准，而且也不是经由WHATWG（参见1.1.3节）制定的。地理定位可以让我们获得与访客位置相关的信息：反映访客当前位置的地理坐标。
- **Web Worker**。随着Web开发人员编写大量JavaScript来武装网页，把一些任务放在后台执行变更越来越重要。这些任务可能要花相对较长时间才能完成，而在后台静静地、无干扰地运行它们已经成为大势所趋。尽管也可以使用计时器和其他技术，但Web Worker规范提供的执行后台任务的方案更简洁。
- **会话历史**。Web刚刚诞生的时候，网页只有一个功能：显示内容。因此，人们把时间大都花在了单击链接，从一个文档跳到另一个文档上。而今天，JavaScript能让网页不必刷新即可加载另一个页面的内容。如此一来，也就创造出了更加流畅的体验。可是，这样也导致了一些问题，比如怎么让浏览器URL与当前内容同步。为了解决这个问题，Web开发人员想出了各种办法，HTML5则为此提供了新的会话历史机制。

13

注意 在探索这三个功能期间，读者会更深入地理解现在所谓的HTML5都包含什么。实际上，很多东西最初都只是一个好想法，后来被整合到这个雄心勃勃的标准中，然后不断充实完善，直至包含很多解决不同问题的新功能，而所有这些功能无非都源于那么几个核心概念（比如语义化、JavaScript和CSS3）。

13.1 地理定位

Geolocation可以让我们确定访客在地球的什么位置。注意，这可不是说判断用户在哪个国家或者哪个城市，而是说确定用户在城市的哪条街道上，甚至是用户正拿着智能手机在哪里上网，

给出这个地点的坐标位置。

注意 本书介绍的大多数JavaScript功能，最初都源自HTML5规范，W3C接手后，把这些规范逐一地分离了出来。但Geolocation不是这样的，它一开始就不属于HTML5，而是与HTML5几乎同时成长起来的。不过，很多人到现在仍然把它和HTML5混为一谈，把它们当成同一类Web新技术。

13.1.1　地理定位的基本原理

只要不是神经有问题，人们都会提出类似这样的问题：几行代码就能确定我现在在哪个咖啡馆里面？是不是有什么隐藏的程序在跟踪我呀？屋外的白色面包里车里坐的是什么人？

放心吧，地理定位跟“黑社会老大”真扯不上半点关系。因为就算浏览器支持地理定位，如果你不允许，它也不会把你的行踪透露给浏览器（参见图13-1）。

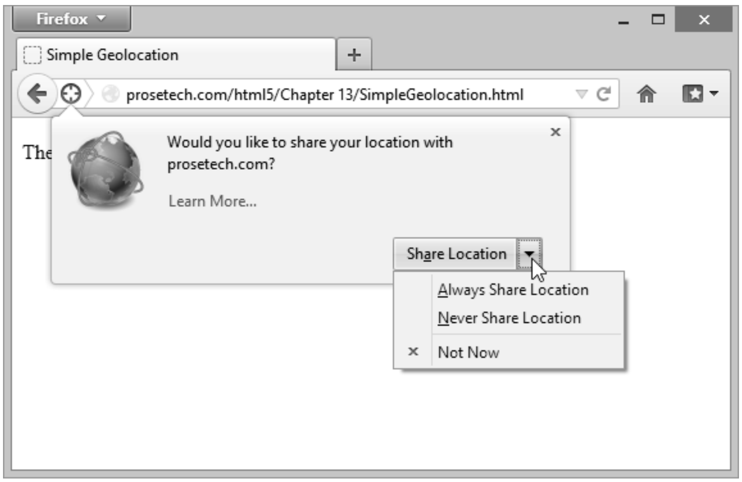


图13-1：网页正想加载位置数据，Firefox询问你是允许一次（单击Share Location），还是以后都允许（Always Share），抑或永远不允许（Never Share）。这可不是Firefox有礼貌，而是Geolocation标准要求浏览器必须征得用户同意，才能允许网站访问其数据

为了得到用户的位置信息，浏览器会争取位置提供商（location provider）的帮助。比如，Firefox浏览器使用的是Google Location Services。位置提供商为查找位置要付出很大努力，而且要想尽各种办法。

对于通过网线（不是无线）上网的桌面计算机来说，办法很简单，但结果不太准确。用户一上网，他的信息就会通过双绞线在计算机或本地网上传输，进入电话线，或者拨号连接（令人恐惧），最终到达连接因特网的高层硬件设备。这个设备有一个唯一的IP地址，靠这个就能在因特网上找到它。与IP地址对应，还有一个现实中的邮政地址。

注意　如果读者对网络知识比较熟悉，那肯定知道自己的计算机与其他计算机一样，都有自己的IP地址。不过，这个IP地址只是你自己私有的，目的是区分你的计算机与同一网络中的其他计算机（比如厨房里的笔记本或背包里的平板）。地理定位不使用私有IP地址。

位置提供商会把这两个信息综合起来。首先，它找到你连接的IP地址，然后，确定使用该IP地址的路由器的位置。因为这个信息是间接的，所以使用桌面计算机时的地理定位并不准确。比如，你在芝加哥西郊的某个地方上网，而你的位置可能会出现在离市中心不远的地方。不过，即便是如此不准确的结果也还是有用的。如果你正在某个地图应用上寻找附近的披萨店，那你可以一下就跳到自己感兴趣的区域——你们家附近，即使离你家还有一段距离。

注意　IP地址定位是最粗略的地理定位方法。如果还有更好的数据源，位置提供商会择优选用。

如果你在使用笔记本或移动设备无线上网，那位置提供商会寻找你附近的无线接入点。理想情况下，位置提供商会查询一个大型数据库，以确定你周围几个接入点的确切位置，然后再使用三角测量法算出你的位置。

要是你在用手机上网，位置提供商还会采用类似的三角测量法，但使用的是信号发射塔的位置。经过迅速而相对准确的计算，最终得到的位置误差大约在1000米。（中心城区等繁华地带的信号发射塔相对较多，因此地理定位的结果也更准确。）

最后，很多移动设备都配有专用的GPS组件。GPS可以使用卫星定位，误差只有几米。但GPS的缺点是速度慢，耗电多。而且，GPS在高楼林立的地区也不好使，因为高大的建筑物会屏蔽信号。当然，到底用不用GPS完全取决于你自己，这一点将在后面介绍（参见13.1.4节）。

当然了，还有其他技术可以用于地理定位。位置提供商会想更多办法获得位置信息，比如RFID芯片、蓝牙设备，以及由Google Maps设置的cookie等。

提示　还可以使用另一个工具来修改自己的起始位置。比如，Chrome迷们可以用一个叫Manual Geolocation（http://tinyurl.com/manual-geo）的插件，它可以在网站想共享你的位置时，告诉Chrome提供哪里的位置信息。利用这个工具甚至可以伪造位置，比如假装自己在德国洛瓦，而实际上是在荷兰。这种技术不只是用来做间谍活动，对调试有定位需求的Web App也非常有用。

关键在于，无论你通过什么方式上网——就算是你使用台式机，地理定位都可以大概地找到你。而如果你使用能接收电话信号或者配有GPS芯片的设备，地理定位的坐标将惊人地精确。

怎么使用地理定位

在理解了第一个大问题——地理定位的工作原理之后，接下来就要面临第二个问题，即为什么要使用地理定位？

要回答这个问题，必须明白地理定位返回的结果是一个人所在位置的坐标，仅此而已。你需要利用这个简单但却基本的信息，进一步取得更详细的位置数据。这些数据可能来自 Web 服务器（一般都要涉及访问大型的服务器端数据库），或者其他地理位置 Web 服务（比如 Google Maps）。

举个例子，如果你在现实中有一家公司，那么可能要将访客的位置与你自己的位置进行比较。然后才知道哪个位置最近。再比如，要是你正在开发一款社交应用，那么可以获得一群人的位置信息，显示他们彼此距离有多近。当然，还可以根据用户的位置信息，向他们推荐一些服务，比如布鲁克林最近的巧克力店，或者最近的厕所。无论如何，访客的地理位置坐标，只有在跟更多的地理数据结合起来时才有意义。

13.1.5 节将以 Google Maps 为例介绍地理定位的应用。至于其他公司的地图应用或地理位置服务，我们就不多介绍了。

13.1.2 查找访客的坐标

地理定位功能实际上是非常简单的。说到底，主要就是navigator.geolocation对象的三个方法：getCurrentPosition()、watchPosition()和clearWatch()。

注意 可能有读者对navigator对象不熟悉，它只是JavaScript众多对象中的一个，它的属性中保存着当前浏览器及其能力的信息。其中，最有用的属性莫过于navigator.userAgent，包含了关于浏览器的所有细节，如版本号、所在操作系统等。

要取得访客的位置，可以调用getCurrentPosition()方法。当然，查找位置不会立即返回结果，浏览器也不想锁定页面等待位置数据。所以，getCurrentPosition()方法是异步的，它会立即执行，但不会阻塞其他代码。完成地理定位后，它会触发另一段代码来处理返回的结果。

噢，地理定位会通过一个事件来告诉我们它完成了工作，这与图片加载完毕或文本文件读取完毕时触发onLoad事件类似，对吗？你错了，JavaScript在这里有点不一致。换句话说，在调用getCurrentPosition()时，你要提供一个完成函数。

下面就是一个例子：

```
navigator.geolocation.getCurrentPosition(
  function(position) {
    alert("You were last spotted at (" + position.coords.latitude +
     "," + position.coords.longitude + ")");
  }
);
```

运行这段代码，会调用getCurrentPosition()并传给它一个函数。浏览器确定了位置之后，会触发传入的函数，然后显示一个对话框。图13-2展示了Internet Explorer中看到的结果。

为了让代码更清晰，可以把完成函数的定义挪到getCurrentPosition()调用语句的外面来，如下所示：

```
function geolocationSuccess(position) {
  alert("You were last spotted at (" + position.coords.latitude +
   "," + position.coords.longitude + ")");
}
```



图13-2：上：首先，要同意浏览器向Web服务器透露你的位置信息。下：结果是你在地球上的坐标

然后，在调用getCurrentPosition()时再传入函数名即可：

```
navigator.geolocation.getCurrentPosition(geolocationSuccess);
```

测试的时候，需要使用支持地理定位的浏览器，并且允许网页访问你的数据。另外，建议你把页面上传到测试服务器，然后再通过浏览器打开。否则，有可能遇到奇怪的现象（例如，地理定位无法处理错误），而有些浏览器（比如Chrome）根本不会检测你的位置。

“地理坐标对我有什么用吗？”问得好。我们马上会介绍怎么利用这个地理数据（见13.1.5节）。但在此之前，有必要先了解一些错误处理和地理定位设置的知识。

确定地理定位的精确程度

调用的 getCurrentPosition()成功返回后，position 对象会包含两个属性：timestamp（确定地理位置的时间）和 coords（地理坐标）。

从上面的例子可以看到，coords 属性是一个对象，包含 latitude（纬度）和 longitude（经

度）属性，用以确定你在地球上的位置。不过，coords 属性还有另外一些信息，比如 altitude（海拔高度）、heading（移动方向）和 speed（移动速度），但目前尚未有浏览器支持它们。

更有意思的是 accuracy（精度）属性，用米为单位表示地理定位信息的精确度。（不要搞混，accuracy 属性的值越小，表示地理定位越精确。）例如，accuracy 等于 2135 米，约 1.3 英里，表示地理定位在该范围内找到了当前访客的位置。为了好理解，可以想象一个圆，圆心是返回的地理坐标，半径是 2135 米，而访客可能就在这个圆形区域中的某个地方。

利用 accuracy 属性可以确定哪些地理定位结果不能用。比如，一个结果的精确度为几万米，那跟没有定位也差不多了。

```
if (position.coords.accuracy > 50000) {
  results.innerHTML =
    "This guess is all over the map.";
}
```

此时，可能需要通知用户无法准确定位，或者为用户提供一个文本框，让他自己输入位置信息。

13.1.3　处理错误

要是访客不愿意共享他们的位置数据，那地理定位就不会返回位置信息。在这种情况下，根本不会调用完成函数，而页面也没有办法告诉你浏览器是在继续挖掘数据，还是遇到了错误。为了解决这个问题，可以在调用getCurrentPosition()时传入两个函数。第一个函数在页面成功取得数据时调用，第二个函数在地理定位因错误而终止时调用。

下面就是一个同时传入完成函数和错误函数的例子：

```
// 保存显示结果的页面元素
var results;

window.onload = function() {
  results = document.getElementById("results");

  // 如果浏览器支持地理定位，取得访客的位置
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      geolocationSuccess, geolocationFailure
    );
    results.innerHTML = "The search has begun.";
  }
  else {
    results.innerHTML = "This browser doesn't support geolocation.";
  }
};

function geolocationSuccess(position) {
  results.innerHTML = "You were last spotted at (" +
   position.coords.latitude + "," + position.coords.longitude + ")";
}
```

```
function geolocationFailure(positionError) {
  results.innerHTML = "Geolocation failed.";
}
```

调用错误函数时，浏览器会给错误函数传入一个错误对象，这个对象有两个属性：code和message。其中，code属性是一个数值，表示问题类型；而message中包含着对问题的简短描述。一般来说，message属性多用于测试，而code属性用于确定如何进行下一步的处理。

下面是修改后的错误函数，检测了code属性所有可能的值：

```
function geolocationFailure(positionError) {
  if (positionError.code == 1) {
    results.innerHTML =
     "You decided not to share, but that's OK. We won't ask again.";
  }
  else if (positionError.code == 2) {
    results.innerHTML =
     "The network is down or the positioning service can't be reached.";
  }
  else if (positionError.code == 3) {
    results.innerHTML =
     "The attempt timed out before it could get the location data.";
  }
  else {
    results.innerHTML =
     "This the mystery error. We don't know what happened.";
  }
}
```

注意 如果你是在本地计算机（而非真正的Web服务器）上测试页面，那么在拒绝共享位置信息后，不会触发错误函数。

13.1.4 设置地理定位选项

到目前为止，我们已经知道调用getCurrentPosition()时可以传入两个参数：一个成功函数，一个失败函数。实际上，还可以再传入一个参数，这个参数是一个对象，用于设置某些地理定位选项。

可以设置三个选项，每个选项对应地理定位选项对象的一个不同的属性。这三个选项可以只设置一个，也可以设置多个。下面这个例子设置了名为enableHighAccuracy的选项：

```
navigator.geolocation.getCurrentPosition(geolocationSuccess,
 geolocationFailure, {enableHighAccuracy: true});
```

下面这个例子则设置了三个选项：

```
navigator.geolocation.getCurrentPosition(
 geolocationSuccess, geolocationFailure,{enableHighAccuracy: true,
 timeout: 10000,
```

```
  maximumAge: 60000}
);
```

这两个例子都使用JavaScript对象字面量设置了地理定位选项。要完美地运行这种技术，就要使用正确的属性名，比如enableHighAccuracy和timeout，因为这些是getCurrentPosition()方法要求的属性。（如果这段代码对你来说还是有点难懂，可以参考附录B中与对象相关的介绍。）

好了，这些属性都是什么意思呢？enableHighAccuracy属性要求高精度的GPS位置检测，只要设置支持（而且用户同意）。除非确实需要精确的坐标，否则不要设置这个选项，因为这个过程很费电也很耗时。而enableHighAccuracy的默认值，也就是不设置它时的值，是false。

第二个timeout属性用于设置在最终放弃之前，等待位置数据的时间，以毫秒计。这里的10 000毫秒的意思就是最多等10秒钟。用户按下同意共享数据的按钮时，计时开始。默认情况下，timeout的值是0，也就是页面会无限期地等下去，不会触发超时错误。

最后一个maximumAge属性用于缓存位置数据。比如，把maximunAge设置为60 000毫秒，之前的数据最多保存1分钟。这样就可以减少重复调用地理定位的次数，但在用户移动的时候，时间越长结果也就越不精确。默认情况下，maximumAge的值是0，意思是不使用缓存的地理数据。（也可以将其设置为一个特殊的值Infinity，表示只要有缓存的数据，就使用该数据，无论时间多久都行。）

13.1.5　显示地图

取得用户的地理坐标当然值得高兴，但除非你能用它实现更有用的功能，否则这种兴奋劲也保持不了多久。精通地理定位的老手深知，通过位置信息可以挖掘出很有价值的东西。（这里的关键在于取得坐标，然后在Web应用中将其转换为一种有用的形式。）我们知道，有很多不错的Web应用，比如Google Maps。事实上据估计，Google Maps是人们日常生活中使用率最高的Web应用服务，而使用它的目的千差万别。

比如，可以使用Google Maps创建一个地图，以任何大小显示世界上的任何一个地方。你可以控制访客能对地图进行什么操作，生成行车线路，而最有用的则是把自定义的数据点放到地图上面。比如，可以用Google Maps创建一个页面，显示你们公司的位置，或者标记出徒步到曼哈顿岛旅行必须参观的景点。在使用Google Maps开发之前，请先看看这份文档：http://tinyurl.com/maps-docs。

注意　Google Maps可以免费使用，而且可以在商业网站中使用，只要你不对网站用户收费就行。（如果你对访客收费，谷歌还有付费地图服务，你可以买来使用。）虽然Google Maps在许可条款中明确表示保留在地图上显示广告的权利，但至今还没有显示广告。

图13-3展示了前面地理定位页面的增强版，即取得用户坐标后，在地图上显示出他的位置。

创建这个页面很简单。首先，链接到Google Mpas API脚本，而且要把它放在使用地图功能的自定义脚本前头：

```
<head>
  <meta charset="utf-8">
  <title>Geolocation Map</title>
  <script src="http://maps.google.com/maps/api/js?sensor=true"></script>
  ...
</head>
```

然后，需要一个<div>元素，让它盛放动态生成的地图。为了方便引用，还要给它一个ID：

```
<body>
  <p id="results">Where do you live?</p>
  <div id="mapSurface"></div>
</body>
```

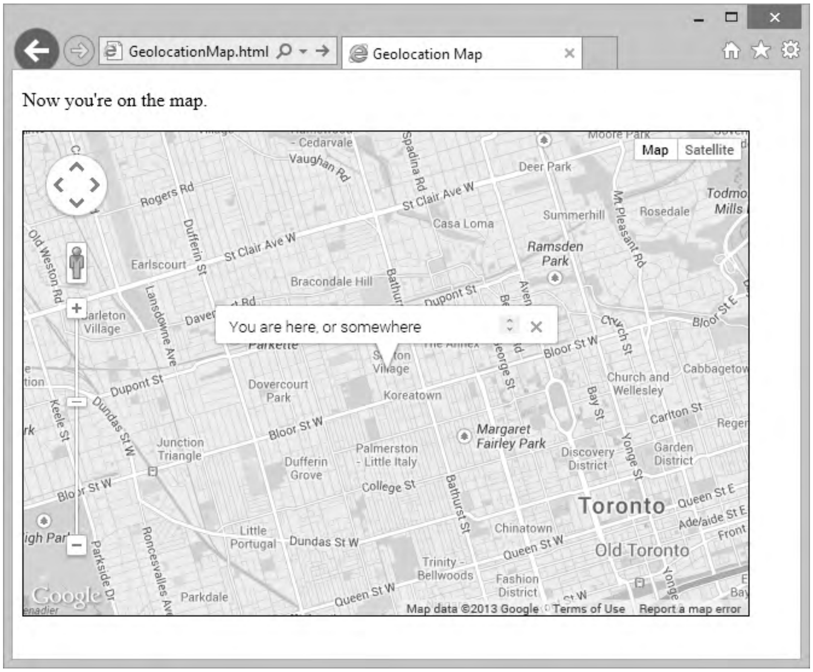


图13-3：地理定位和Google Maps是一个强有力的组合。只需几行JavaScript代码，就可以生成任何位置的地图

这样就可以给它设置样式，声明地图的大小：

```
#mapSurface {
  width: 600px;
  height: 400px;
  border: solid 1px black;
}
```

使用Google Maps的准备工作就做好了。接下来首先要考虑显示地图。我们这个例子是在页面加载时就显示了地图，因此要使用成功或失败函数。（执行失败函数并不意味着访客不能在你的页面中使用地图，而只是说你没有取得访客的当前位置。这时候还是可以显示地图，只不过显示的是默认位置。）

以下就是页面加载时运行的代码，首先创建地图，然后通过地理定位查找用户位置：

```
var results;
var map;

window.onload = function() {
  results = document.getElementById("results");

  //设置地图选项。这个例子设置了起始缩放级别和地图类型
  //要了解所有可用的选项，读者可以查看Google Maps的文档
  var myOptions = {
    zoom: 13,
    mapTypeId: google.maps.MapTypeId.ROADMAP };

  //使用前面设置的选项来创建地图
  map = new google.maps.Map(document.getElementById("mapSurface"), myOptions);

  //尝试取得用户的当前位置
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(geolocationSuccess,
     geolocationFailure);
    results.innerHTML = "The search has begun.";
  }
  else {
    results.innerHTML = "This browser doesn't support geolocation.";
    goToDefaultLocation();
  }
};
```

即便通过以上代码创建了地图，还不能在页面上看到它。因为我们还没有设置地理位置呢。要设置地理位置，得用LatLng对象创建一个坐标点，然后再通过地图的setCenter()方法把该点放到地图上。以下就是使用访客坐标创建坐标点并将该点放到地图上的代码：

```
function geolocationSuccess(position) {
  //把地理定位的位置转换为LatLng对象
  location = new google.maps.LatLng(
   position.coords.latitude, position.coords.longitude);

  //在地图上显示这个点的位置
  map.setCenter(location);
```

有了这些代码就可以显示地图了，结果如图13-3所示。除此之外，还可以在地图上添加一些辅助功能，比如其他位置或信息气泡。就拿创建信息气泡为例，需要创建一个InfoWindow对象。而图13-3中的信息气泡就是用下面的代码创建的：

```
  //创建信息气泡并设置其文本内容和地图坐标
  var infowindow = new google.maps.InfoWindow();
  infowindow.setContent("You are here, or somewhere thereabouts.");
  infowindow.setPosition(location);

  //显示地图气泡
  infowindow.open(map);
```

```
    results.innerHTML = "Now you're on the map.";
  }
```

最后，如果浏览器不支持地理定位，那么处理方法也是类似的。只不过是使用一个你自己知道的坐标来创建地图而已。

```
function geolocationFailure(positionError) {
  ...
  goToDefaultLocation();
}

function goToDefaultLocation() {
  //这里显示纽约地图
  var newYork = new google.maps.LatLng(40.69847, -73.95144);

  map.setCenter(newYork);
}
```

13.1.6 跟踪访客移动

我们的例子一直在使用getCurrentPosition()方法，这个方法可以说是地理定位的“心脏”。而除这个方法之外，地理定位对象还有两个方法，用于跟踪访客的位置，让你的页面在用户位置改变时能收到通知。

首先是watchPosition()方法，它与getCurrentPosition()看起来极为相似，也接收三个参数：成功函数（唯一必需的参数）、失败函数和选项对象：

```
navigator.geolocation.watchPosition(geolocationSuccess, geolocationFailure);
```

但watchPosition()与getCurrentPosition()的区别在于，前者可能会多次触发成功函数——不仅初始取得位置时会触发，而且以后每次检测到新位置都会触发。（但你无法控制设备多长时间检测一次新位置。你只要知道，位置不改变，设备不会给你发通知，只有位置改变它才会给你发通知。）对桌面计算机而言，因为它不会动，所以watchPosition()与getCurrentPosition()方法实际上作用相同。

与getCurrentPosition()不同，watchPosition()返回一个数值。如果你不想再关注位置变化，可以把它返回的这个数值传给clearWatch()方法。当然，你也可以不这么做，而在用户切换到其他页面之前一直接收通知：

```
var watch = navigator.geolocation.watchPosition(geolocationSuccess,
 geolocationFailure);
...

navigator.geolocation.clearWatch(watch);
```

13.1.7 浏览器对地理定位的兼容情况

所有的现代浏览器都能很好地支持地理定位功能，包括移动浏览器。旧版的IE是个例外——

IE7和IE8还不支持地理定位。想要照顾到旧版IE的用户，可以填补这个空缺。有很多可以解决这个问题的简单JavaScript库。通常，它们会用到13.1.1节介绍的IP查找技术，这是最简陋的地理定位方式。比如，http://github.com/inexorabletash/polyfill上的地理定位“腻子脚本”就是抓取路由器的IP地址，然后在http://freegeoip.net的数据库里查找它的地理位置。

也可以设一个默认的起始位置，而不用获取用户的当前位置。如果是用谷歌地图的话，可以让用户从地图上选取一个点，然后用这个点的坐标。Google Maps API的文档里有一些这样的例子，http://tinyurl.com/qbmqdsq就是一个拦截地图点击事件的例子。

13.2　Web Worker

想当初，JavaScript刚刚问世的时候，没有人担心它的性能。JavaScript只是一种简单的语言，可以在网页中运行小段脚本，而且只是非专业程序员的玩具。谁也没有把它当做一门能撑起门面的正规语言。

转眼20年过去了，JavaScript已经成为Web开发领域的王者。只要想给网页添加交互性，开发人员就会用到它，无论是游戏还是地图，或者购物车和漂亮的表单。然而，从许多方面来看，JavaScript语言与它现在的地位相比，仍然还有一些不相称的地方。

比如，JavaScript处理大计算量任务时就会导致问题。对于多种现代编程语言来说，这种大计算量的工作都是在后台完成的，而使用应用的人不会停下来，也不会受到干扰。但在JavaScript中，由于代码始终都在前台运行，因此耗费时间的代码会打断用户，阻塞页面，直到任务完成。对这个问题视而不见，就会导致访客厌烦，甚至永不再光顾你的网页。

注意　为解决JavaScript阻塞页面的问题，很多一线开发人员想出了各种招。比如，使用`setInterval()`或`setTimeout()`把大任务分成小任务，每次只运行一个小任务。这个办法非常适合某些任务（比如，对在Canvas上实现动画就很合适，参见9.4节）。可是，对于不能拆分而又耗时很长的任务，这个办法会增加复杂性和困扰。

HTML5提出了更好的解决方案，一个叫Web Worker的对象，能够在后台完成工作。要是你有比较费时的工作，就可以创建一个新的Web Worker对象，把要运行的代码交给他，然后让它运行就好了。在它工作期间，还可以通过传递文本消息这种安全但受限的方式与它通信。

Web Worker安全措施

在 JavaScript 中使用 Web Worker 可以在后台运行代码，同时在前台也做一些事。这就带来了编程领域中的一个尽人皆知的问题：如果应用同时可以做两件事，那么其中一件事有可能干扰另一件事。

这个问题会在两段代码争抢同一处数据时发生。比如，其中一段代码想读取某些数据，而另一段代码则想写入该数据，或者两段代码同时想设置一个变量，最终导致一个值覆盖另一个

值，再或者两段代码以不同方式操作同一个对象，造成对象状态前后不一致。类似的问题很多，没办法一一列举和解决。通常，一个多线程应用（即在多个**线程**上执行不同代码的应用）在测试时运行得很好，但一投入正常使用，就会出现令人头疼的数据不一致问题。

现在好了，有了 JavaScript 的 Web Worker，你就不必担心这个问题了。因为它不允许你在网页之间或 Web Worker 之间共享数据。你可以把**数据**从网页发送到 Web Worker（或者相反），但 JavaScript 会自动复制一份，并发送该副本。这意味着不同的线程不能同时占用相同的内存区域，也不会导致微妙的问题。当然，这种简化的模型也会限制 Web Worker 的能力，但能力上受到细微的限制却能换来安全，免得那些编程高手搬起石头砸自己的脚。

13.2.1 费时的任务

除非用于那些真正费时的任务，否则很难发挥Web Worker的优势。换句话说，不应该用Web Worker来执行简单的任务。而对于那些让CPU不堪重负，又会拖延浏览器的计算任务，使用Web Worker的结果会大不相同。比如图13-4所示的搜索素数的任务，我们想找到某个区间内的素数。代码很简单，但这个任务需要的计算量很大，因为要进行较长时间的数值运算。

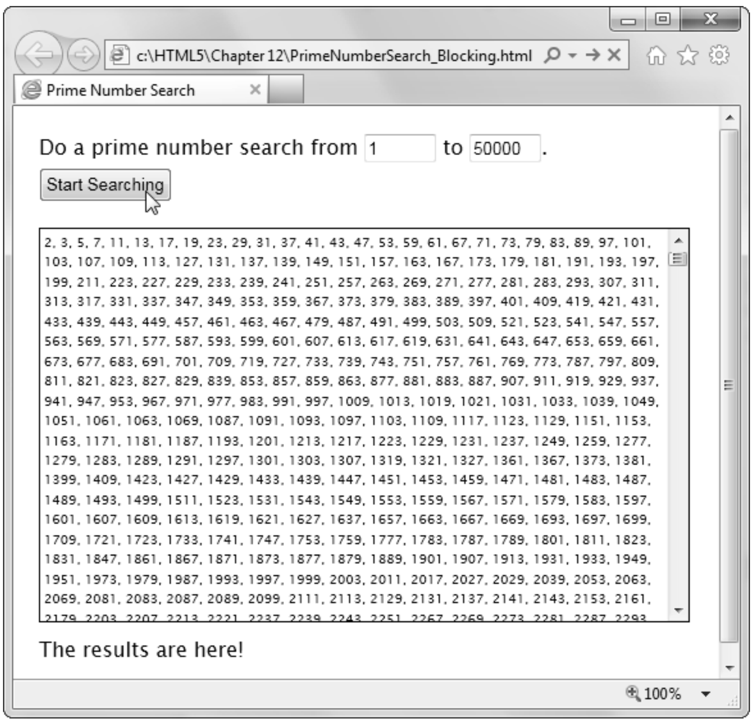


图13-4：选择一个区间，然后单击按钮开始搜索。区间窄（如1～50 000），任务很快能完成，不会干扰任何人。但范围更大的搜索（如1～500 000），会导致页面数分钟没有反应。这时候不能单击、滚动，无法执行任何操作。浏览器甚至会给出一个“长时间运行脚本”的警告，并将整个页面灰掉

13

很明显，可以使用Web Worker来改进这个页面。但在此之前，我们先看一看这个例子的标记和JavaScript代码。

标记不长，也很简单。页面使用了两个<input>控件，都是文本框。还有一个用于搜索的按钮和两个<div>元素，两个<div>分别用于保存结果和显示状态消息。以下就是<body>元素中的全部标记：

```
<p>Do a prime number search from <input id="from" value="1"> to
 <input id="to" value="20000">.</p>
<button id="searchButton" onclick="doSearch()">Start Searching</button>

<div id="primeContainer">
</div>

<div id="status"></div>
```

给保存素数的<div>元素添加样式有点意思。我们给它指定了固定高度和一个最大宽度，还设置了overflow和overflow-x属性以添加垂直滚动条（但没有水平滚动条）：

```
#primeContainer {
  border: solid 1px black;
  margin-top: 20px;
  margin-bottom: 10px;
  padding: 3px;
  height: 300px;
  max-width: 500px;
  overflow: scroll;
  overflow-x: hidden;
  font-size: x-small;
}
```

这个例子的JavaScript代码有点长，可并不复杂。代码会取得文本框中的两个数，开始搜索，然后把找到的素数添加到页面中。查找素数的任务是由另一个函数完成的，该函数名叫findPrimes()，而且保存在另一个JavaScript文件中。

提示 要理解这个例子或者Web Worker的作用，不必去看findPrimes()函数，只要知道这是个费时间的任务就行了。不过，假如你对这里的数学运算感兴趣，或者想自己写一个搜索素数的脚本，也可以在本书站点上找到该函数的完整代码：http://prosetech.com/html5。

下面是doSearch()函数的完整代码：

```
function doSearch() {
  //取得指定搜索区间的两个数
  var fromNumber = document.getElementById("from").value;
  var toNumber = document.getElementById("to").value;

  //执行搜索（这一步花时间）
  var primes = findPrimes(fromNumber, toNumber);
```

```
    //遍历素数数组，把它们转换成一个长字符串
    var primeList = "";
    for (var i=0; i<primes.length; i++) {
      primeList += primes[i];
      if (i != primes.length-1) primeList += ", ";
    }

    //把素数字符串插入页面中
    var displayList = document.getElementById("primeContainer");
    displayList.innerHTML = primeList;

    //更新状态消息，告诉用户当前情况
    var statusDisplay = document.getElementById("status");
    if (primeList.length == 0) {
      statusDisplay.innerHTML = "Search failed to find any results.";
    }
    else {
      statusDisplay.innerHTML = "The results are here!";
    }
  }
```

看到了吧，标记和代码都不长，实际上可以说是简明扼要。但这只是表面上能看到的，如果搜索的区间很大，那搜索过程会得变慢吞吞地无法忍受，就好像开着高尔夫球车爬陡坡一样费劲。

13.2.2 把任务放在后台

Web Worker为解决这个问题定义了一个新对象，叫Worker。在需要在后台执行任务时，可以创建一个新的Worker，交给它一些代码，然后发送给它一些数据。

下面这行代码创建了一个新的Worker对象，让它执行PrimeWorker.js中的代码：

```
var worker = new Worker("PrimeWorker.js");
```

让Worker运行的代码都要放在一个单独的文件中。这样设计是为了避免新手让Worker引用全局变量，或者直接访问页面中的元素。这两个操作都是不允许的。

13

注意 浏览器会严格保持网页与Web Worker代码分离。因此，不可能让PrimeWorker.js中的代码把素数直接写到`<div>`元素里。Web Worker必须把相应数据发送给页面中的JavaScript代码，然后再通过它把结果显示出来。

网页与Web Worker之间通过消息来沟通。给Worker发送消息要使用该对象的`postMessage()`方法：

```
worker.postMessage(myData);
```

然后，Worker就会通过`onMessage`事件接收到该数据的一个副本。此时，它就开始工作。

类似地，如果Worker需要跟网页对话，它可以调用自己的`postMessage()`方法，并带上一些数据。而网页同样是在一个`onMessage`事件中接收这些数据。图13-5展示了上述通信过程。

我们先说一个要注意的地方。调用`postMessage()`方法时，只能给它传入一个值。这对于传

递搜索素数的区间来说是一个问题，因为区间由两个值确定。我们的方案是把这两个值放到一个对象字面量中（参见附录B）。下面的代码是一个例子，这里的对象字面量包含两个属性（第一个是from，第二个是to），每个属性都有一个值：

```
worker.postMessage(
 { from: 1,
   to: 20000 }
);
```

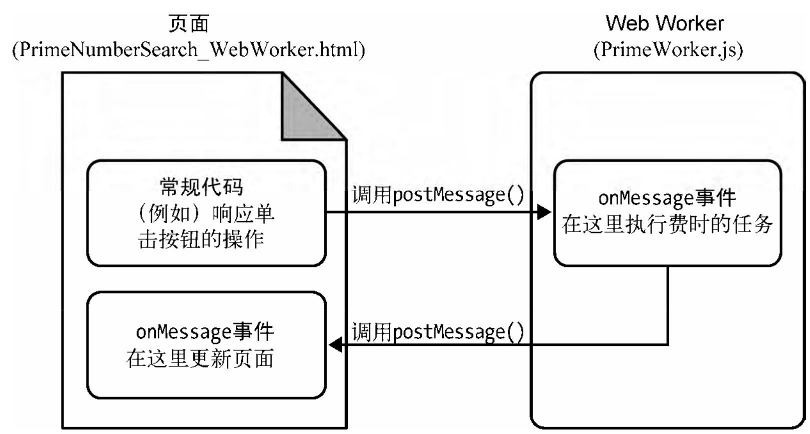


图13-5：这里演示了最简单的Web Worker工作流程，分三个步骤：页面向Worker发送一些数据，Worker开始运行，然后Worker再向页面发回一些数据

注意 请注意，你可以给Worker传入任何对象字面量。到了后台，浏览器会使用JSON（参见10.2.4节）将传入的对象转换为无害字符串，复制它，然后再重新将其转换成对象。

了解了这些细节后，就可以对前面看到的doSearch()函数进行一番改进了。这里，不再让它自己搜索素数，而让它创建一个Worker来承担相应的任务：

```
var worker;

function doSearch() {
  //禁用按钮，防止用户同时输入多个搜索区间
  searchButton.disabled = true;

  //创建新的Worker
  worker = new Worker("PrimeWorker.js");

  //指定onMessage事件
  //以便从Worker那里收到消息
  worker.onmessage = receivedWorkerMessage;

  //取得数值范围，发送给Web Worker
  var fromNumber = document.getElementById("from").value;
  var toNumber = document.getElementById("to").value;
```

```
    worker.postMessage(
     { from: fromNumber,
      to: toNumber }
    );

    //告诉用户正在搜索
    statusDisplay.innerHTML = "A web worker is on the job ("+
     fromNumber + " to " + toNumber + ") ...";
  }
```

现在，PrimeWorker.js开始工作了。它接收到onMessage事件，执行搜索，然后给网页发送回一条新消息，包含找到的素数：

```
onmessage = function(event) {
  //网页发过来的对象保存在event .data属性中
  var fromNumber = event.data.from;
  var toNumber = event.data.to;

  //在该数值范围内搜索素数
  var primes = findPrimes(fromNumber, toNumber);

  //搜索完成，把结果发回网页
  postMessage(primes);
};

function findPrimes(fromNumber, toNumber) {
  //（费事的素数判断过程都在这个函数里）
}
```

在Worker调用postMessage()的时候，就会触发onMessage事件，进而会调用网页中的receivedWorkerMessage()函数：

```
function receivedWorkerMessage(event) {
  //取得素数列表
  var primes = event.data;

  //把素数显示出来
  ...

  //启动搜索功能
  searchButton.disabled = false;
}
```

这里省略的代码与在上一节看到的一样，就是把素数的数组转换为文本，然后把文本插入到网页中。

总的来说，代码的结构是变了，但逻辑相差无几。可是，结果呢？完全不一样。现在，即便搜索大量的素数，页面也能保持响应。可以滚动页面、在文本框里输入、选择之前搜索的结果。要不是页面底部的消息，恐怕没人会知道是后台使用了Web Worker。

提示 你的Web Worker需要使用另一个JavaScript文件中的代码吗？可以使用`importScripts()`函数。假如你需要在PrimeWorker.js中调用FindPrimes.js文件中的函数，只要添加下面这行代码即可：

```
importScripts("FindPrimes.js");
```

离线运行Web Worker

如果花点时间准备一个完整的 Web Worker 的例子并将它上传到服务器，测试页面就没那么复杂。当然，在现实世界里，在桌面上试验并且直接从本地硬盘运行还是简单许多。这种方式在 Firefox 和当前版本的 IE 上有效，但在 Chrome 上无效，除非做一点额外的工作。

如果是从本地打开 Web Worker 页面，Chrome 需要在启动时带上参数`--allowfile-access-from-files`。在 Windows 中，这可以通过修改 Chrome 快捷方式（右键点击并选择属性）来实现，或创建一个新的 Chrome 快捷方式来配置。不论采用哪种方式，都需要将参数附加到命令行的后面。比如，像下面这个快捷方式路径：

```
C:\Users\billcruft\ ... \chrome.exe
```

要修改为：

```
C:\Users\billcruft\ ... \chrome.exe
--allow-file-access-from-files
```

现在就可以在舒适又私密的个人电脑上调试 Web Worker 了，不需要上传到服务器。

13.2.3　处理Worker错误

我们知道，`postMessage()`方法是跟Web Worker通信的关键。不过，还有一种方式可以给网页发送通知——用`onerror`事件告诉网页有错误发生：

```
worker.onerror = workerError;
```

这样，如果后台脚本遇到了问题或因为数据无效出现错误，Worker就能把打包的错误数据发送给网页。以下就是一个在网页中显示错误消息的示例代码：

```
function workerError(error) {
  statusDisplay.innerHTML = error.message;
}
```

除了`message`属性外，错误对象还有`lineno`和`filename`属性，分别保存着错误所在的行号及文件的名字。

13.2.4　取消后台任务

学习了一个简单的Web Worker的用例之后，下面该考虑如何改进了。首先，要支持取消后台任务，也就是让网页能在任务执行期间中断它。

停止Worker工作的方式有两种，一种是Worker对象调用自己的`close()`方法，但更常用的是创建Worker对象的页面调用该对象的`terminate()`方法。比如，下面的代码就可以用来响应取消任务的按钮单击操作：

```
function cancelSearch() {
  worker.terminate();
  statusDisplay.innerHTML = "";
  searchButton.disabled = false;
}
```

单击这个按钮就可以停止当前的搜索任务，重新启用搜索按钮。别忘了，以这种方式停止Worker之后，就不能再给它发送任何消息，也不能让它执行任何操作了。要执行新的搜索任务，必须创建一个新的Worker对象。（现在的例子正是这样做的，所以重新启动操作没有问题。）

13.2.5　传递复杂消息

关于Web Worker，最后我们还想介绍一项技术，那就是进度信息。图13-6展示了一个改进后的示例页面，其中显示了进度信息。

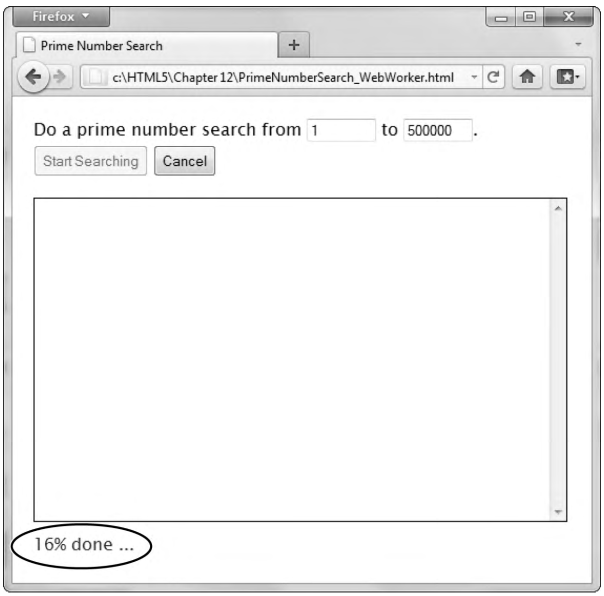


图13-6：在搜索素数的过程中，会不断更新状态消息，告知用户完成了百分之多少。要更好看一点的话，可以像图5-7那样用一个有颜色填充的进度条

要显示进度，Web Worker必须在工作的同时把进度百分比发给页面。我们知道，Web Worker只有一个与创建它的页面对话的方式，即使用`postMessage()`方法。因此，要发送进度百分比，就要发送两种消息：进度通知（在工作过程中）和素数列表（工作结束后）。这项技术的关键在于明确区分两类消息，因此页面中的`onMessage`事件处理程序就知道哪个是进度信息，哪个是结果了。

为此，需要在发送消息对象字面量时定义不同的属性。比如，在Web Worker发送进度信息时，可以将该信息命名为“Progress”，而在发送素数列表时，将其命名为“PrimeList”。

在消息对象字面量里定义新的属性，是我们前面向Web Worker发送区间数值时用过的技巧。这里的新属性messageType和data，分别用于描述消息类型和数据本身。

好了，我们可以重写Web Worker的代码，为素数列表添加一个messageType属性：

```
onmessage = function(event) {
  //搜索素数
  var primes = findPrimes(event.data.from, event.data.to);

  //发回结果
  postMessage(
    {messageType: "PrimeList", data: primes}
  );
};
```

为了发回进度信息，findPrimes()函数里的代码也要调用postMessage()向网页发回信息。这里传递的对象字面量同样包含两个属性：messageType和data，但此时前者声明的是进度通知，后者指定的是进度百分比：

```
function findPrimes(fromNumber, toNumber) {
  ...

  //计算进度百分比
  var progress = Math.round(i/list.length*100);

  //只在进度变化超过1%时才发送进度百分比信息
  if (progress != previousProgress) {
    postMessage(
      {messageType: "Progress", data: progress}
    );
    previousProgress = progress;
  }
  ...
}
```

页面在接收到这个消息后，首先要检查messageType属性，以便知道收到的是什么数据。如果是素数列表，则将结果显示在网页中。如果是进度通知，则更新进度文本：

```
function receivedWorkerMessage(event) {
  var message = event.data;

  if (message.messageType == "PrimeList") {
    var primes = message.data;

    //显示素数列表，这里的代码与前面例子中的一样
    ...
  }
  else if (message.messageType == "Progress") {
    //报告当前进度
    statusDisplay.innerHTML = message.data + "% done ...";
  }
}
```

注意 可以采取另一种方式来设计这个页面。可以让Web Worker每找到一个素数时就调用一次postMessage()方法。这样网页就可以实时地把找到的素数显示出来。显然，此时的优点是可以同步显示结果，但缺点则是不停地阻断页面（因为Web Worker查找素数的速度很快）。到底怎么设计才好呢？这取决于你的任务，比如完成任务的时间长短、只显示部分结果有没有价值、得到每部分结果的效率如何，等等。

利用Web Worker的其他方式

搜索素数的例子是使用 Web Worker 的最直观方式，即执行具有明确描述的任务。每次搜索开始，页面都会创建一个新的 Worker 对象，每个 Web Worker 对象独立负责一项任务。而且，每个对象只接收一个消息，然后发回一个消息。

恐怕实际开发中的页面不会这么简单。下面我们列出一些可能的情况，让你能够进一步扩展这里的例子，进而满足自己的实际需要。

- **在多个任务中重用 Web Woker**。Worker 对象完成既定任务，触发 onMessage 事件处理程序后并不会被销毁。它只会闲置在那儿，等待新的任务。如果你再给它发送新的消息，它会马上进入状态，投入新的工作。
- **创建多个 Web Worker**。一个页面并不限于只能创建一个 Worker 对象。比如，若要支持 访客同时搜索多个区间内的素数，就需要为每个搜索单独创建一个 Worker，然后通过数组来跟踪它们。这样，每当有 Worker 返回结果，就可以把结果添加到页面中，同时注意不覆盖其他 Worker 的结果。（为了稳妥起见，还是建议大家少创建 Web Worker，它们都不是“省油的灯”，一次运行太多会拖慢计算机。）
- **在一个 Web Worker 中创建另一个 Web Worker**。每个 Web Worker 都可以创建自己的 Web Worker，向它们发送消息，从它们那里接收消息。对于复杂的计算任务，比如计算斐波那契数这种需要递归的计算，在 Worker 内创建 Worker 可以派上用场。
- **通过 Web Worker 下载数据**。Web Worker 可以使用 XMLHttpRequest 对象（参见 12.1.1 节）取得新页面，或者向 Web 服务发送请求。取得了所需的信息后，它们可以调用 postMessage()方法，把数据发回页面。
- **利用 Web Worker 执行周期性任务**。与普通网页中的脚本一样，Web Worker 可以调用 setTimeout()或 setInterval()函数。因此，可以通过 Web Worker 来定期检测某个网站是否有新数据。

13.2.6 浏览器对Web Worker的兼容情况

浏览器对Web Worker的支持不及对地理定位的支持那么广泛。表13-1显示了目前的支持情况。

表13-1　浏览器对Web Worker的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android版Chrome
最低版本	10	3.5	3	4	10.6	5	29

那么面对不支持Web Worker的浏览器该怎么办呢？最简单的办法就是把本来要用Web Worker来做的工作放到前台来做：

```
if (window.Worker) {
  // 浏览器支持Web Worker.
  // 那么为什么不创建一个Web Worker并启动呢?
} else {
  // 浏览器不支持Web Worker.
  // 只需调用素数搜索函数，然后等待.
}
```

这种方式不需要写额外的代码，因为素数搜索函数已经写好了，有没有Web Worker都可以调用。但是，如果任务较大，这种方式可能会阻塞浏览器的运行。所以如果用这种策略，比较明智的方式是警告用户（比如，在页面上显示一条消息），告诉他因为浏览器不支持某些功能，计算过程可能会导致页面临时卡顿。

还有一种替代方案（有点繁琐）是用setInterval()或者setTimeout()伪造一个后台任务。比如，可以写一些代码，每隔一段时间测试几个数。一些“腻子脚本”甚至试图使用这种机制（参见http://tinyurl.com/polyfills中Web Worker部分），但这种方式很容易变得混乱。

13.3　历史管理

HTML5添加了会话历史管理功能，作为对JavaScript历史对象的扩展。这个功能听起来简单，但要知道什么时候或者说为什么该用这个功能，却是一个技巧。

什么，你从前一直都没听说过JavaScript中的历史对象？不要紧，到目前为止，这个对象的功能很有限。说白了，以前的历史对象只有一个属性和三个简单的方法。这个属性是length，表示浏览器的历史列表中有多个条记录，这些记录是你在上网期间访问过的页面的列表。下面就是一个使用历史记录的小例子：

```
alert("You have " + history.length +
 " pages in your browser's history list.");
```

历史对象中被用得最多的方法就是back()，能够让访客在浏览器的历史记录中后退一步：

```
history.back();
```

执行这行的效果就如同访客单击了浏览器的“后退”按钮。类似地，调用forward()方法可以前进一步，或者调用go()方法并指定后退或前进的步数。

除非你想要定制后退和前进按钮，否则这些属性和方法没有多大用处。但是，HTML5又在此基础上添加了一些功能，从而让你能实现原来想做但很难做到的事情。新功能的核心是pushState()方法，通过它可以改变浏览器地址栏中的URL，同时不会导致页面刷新。这是一项

非常贴心的技术，特别适合动态加载新内容，同时无阻滞地更新页面的应用。在这种动态页面中，URL与页面内容无法一一对应，会出现第一个页面加载了另一个页面的内容后，其URL仍然保留在浏览器地址栏中的情况。而此时浏览器的收藏夹功能难以反映实际情况。会话历史管理功能为我们提供了解决这个问题的方案。

如果有读者对上面描述的问题不太理解，请稍安勿躁。下一节，我们将举一个能够利用会话历史的例子。

13.3.1 URL问题

上一章，我们介绍了一个有关中国旅游的网页，其中包含一套内建的幻灯片（参见12.1.3节）。在那个页面上，单击Previous和Next按钮，可以加载不同的照片。但该例子最大的亮点是加载每张照片时，不会刷新页面，不会打断用户的注意力，因为它使用了`XMLHttpRequest`对象。

但是，类似这样的动态页面也存在一个广为人知的限制。即使页面因为加载了新页面而有所变化，但浏览器地址栏中的URL却保持不变（图13-7）。



图13-7：这里是ChinaSites.html页面的两个版本，分别加载了不同的幻灯片。但两个页面的URL却完全相同（都是ChinaSites.html）

为了理解这个问题，可以设想你有一个同事小乔在浏览图13-7所示的图片及说明，他看到了不同的风景，但真正打动他的则是第五张幻灯片，那张幻灯片里有一棵许愿树。小乔兴奋地把这张幻灯片加入了书签，然后用电子邮件把URL发给了朋友克莱尔，又在Twitter上向自己的粉丝推荐这张照片（“把纸碟抛到树上，强过向泉水扔硬币，看这里http://…”）。然而，当小乔再次打开书签，克莱尔单击邮件中的链接，或者粉丝们在Twitter上访问小乔推荐的页面时，他们看到的都是第一张幻灯片。这时候，可能有人会失去耐心，等不及翻到第五张幻灯片就离开了，而有人甚至会觉得摸不着头脑——这个链接发错了吧？如果页面中的幻灯片不止5张，问题还会更麻烦；Flickr的一套照片可能多达数十张甚至数百张。

13.3.2 以往的解决方案：hashbang URL

为解决这个问题，很多人采用一种向网页URL末尾添加信息的方式。几年前，Facebook、Twitter和谷歌这些大公司为他们实现的一种叫hashbang的技术兴奋不已，这种技术也极富争议。所谓hashbang，就是在URL末尾加上#!，然后再附加一些信息。看下面这个例子：

```
http://jjtraveltales.com/ChinaSites.html#!/Slide5
```

这种技术的依据是浏览器会把#后面的所有信息当成URL的附加部分。对于上面的URL来说，浏览器会认为你还是在同一个ChinaSites.html页面上，只不过末尾加了一些附加信息。

另一方面，如果JavaScript代码在修改URL时没有使用#符号，结果会怎么样呢：

```
http://jjtraveltales.com/ChinaSites.html/Slide5
```

这样的话，浏览器会马上向Web服务器发送请求，企图下载新的页面。这显然不对。

那怎么实现这个hashbang技术呢？首先，要在页面每次加载新幻灯片时改变URL。（在JavaScript代码中修改`location.href`属性即可。）其次，要在页面首次加载时检测URL，取得附加信息，然后据以从Web服务器动态地获取对应的内容。整个过程加起来经常会搞得人眼花缭乱，好在有现成的JavaScript库可以使用，比如PathJS（https://github.com/mtrpcic/pathjs）。有了这些库，实现hashbang技术就简单多了。

最近，hashbang技术声名狼藉，许多前支持者也完全放弃了它（具体原因见随后的附注栏）。但是，hashbang仍然出现在一些通信繁多的网站，比如Google Groups。

为什么没人喜欢hashbang

近期，hashbang 方案被广泛使用也饱受争议。今天，Web 开发者们决定回退到之前的方案，原因如下：

- **URL 复杂化。**Facebook 就是这个问题的典型。过去，不用浏览太多页面，URL 很快就会被额外信息填满，变成类似这样：http://www.facebook.com/profile.php?id=1586010043#!/pages/Haskell/401573824771。如今，只要浏览器支持，开发人员就可以使用会话历史功能了。
- **缺乏灵活性**。hashbang 页面在 URL 中保存了很多信息。一旦页面的工作方式或者存储信息的方式改变，那么原来的 URL 就完全不能用了，这是很多网站无法访问的主要原因。
- **搜索引擎优化**。搜索引擎会将不同的 hashbang URL 看做一个页面。对于 ChinaSites.html 来说，这意味着搜索引擎不可能索引每一个景点，而是很可能会忽略附加信息。如果有人搜索“china wishing tree”，很可能不会在结果中看到 ChinaSites.html。
- **Cool URL 问题**。Cool URL 就是那些简短、明确——更重要的是——永远不会变的 URL。Web 之父蒂姆·伯纳斯·李在一个页面中（http://www.w3.org/Provider/Style/URI.html）解释了什么是 Cool URL。不管你多么希望自己的内容在网上永存，都改变不了 hashbang URL 难维护的事实，因此不可能成为 Web 发展的选择。

目前，就使用 hashbang 技术时该如何变通这个问题，虽然很多开发人员都有不同的想法，但大多数人都认同 hashbang 只是 Web 开发历史中很短的一个阶段，很快就会被 HTML5 会话历史功能取代。

13.3.3 HTML5 的方案：会话历史

HTML5的会话历史功能为以上URL问题提供了不同的方案。会话历史功能允许开发人员把URL改成任何形式，而不必非要使用滑稽的#号和!号。比如，对于ChinaSites.html页面中的第四张幻灯片，可以把URL修改成这样：

```
http://jjtraveltales.com/ChinaSites4.html
```

在这种情况下，浏览器并不会真的请求名为ChinaSites4.html的页面，而是还在当前页面，但加载新的幻灯片，这正是我们想要的结果。假如访客在历史记录中回退，那么结果也是一样的。比如，访客前进到下一张幻灯片（URL变为ChinaSites5.html），然后单击后退按钮（返回ChinaSites4.html），浏览器仍然会保持在当前页面，但会触发一个事件，以便我们加载匹配的幻灯片，并恢复到页面的正确版本。

听起来不错啊。可是，这个方案也存在一个大问题。如果你想让会话历史如期工作，那么就必须为每个URL都创建一个对应的页面。对这个例子来说，你必须创建ChinaSites1.html、ChinaSites2.html、ChinaSites3.html，等等。这是因为访客可能会直接访问这些页面，可能是过几天又要通过书签打开其中一个页面，手工输入页面的URL，或者是单击邮件中的链接打开其中的页面。大型互联网公司（如Facebook或Flickr）倒是无所谓，因为他们可以利用少量服务器端代码在不同情况下提供相同的幻灯片。但对于小公司或Web开发人员个人来说，这就意味着很大的工作量。有关解决这个问题的方法，请参见13.3.3节的附注栏。

理解了会话历史功能的工作原理后（这是最难的部分），实际使用它就容易多了。事实上，会话历史功能只涉及两个方法和一个事件，都属于history对象。

最重要的方法是pushState()，利用它可以修改网页的URL部分。为了安全起见，不能修改URL的其他部分。（如果允许修改URL的其他部分，那么黑客们就可以轻而易举地伪装别人的网站，比如让人在银行交易表单中用Gmail登录。）

以下就是把URL的网页部分修改为ChinaSites4.html的代码：

```
history.pushState(null, null, "ChinaSites4.html");
```

这里的pushState()方法接收三个参数，第三个最重要，它是出现在浏览器地址栏中的内容。

第一个参数可以是任何数据，只要你认为它适合表示页面的当前状态。利用这个参数，可以让用户通过浏览器历史记录恢复到不同的页面状态。第二个参数是页面标题，显示在浏览器标题栏。所有浏览器目前都忽略这个参数。如果你不想设置状态，也不想设置标题，那么只要像上面代码所示，在这两个参数位置上提供null值即可。

下面，我们就来看看修改ChinaSites.html页面以匹配当前显示状态的代码。这里使用幻灯片的编号表示页面状态。这个细节很重要，稍后在处理onPopState事件时你就会明白：

```
function nextSlide() {
  if (slideNumber == 5) {
    slideNumber = 1;
  } else {
    slideNumber += 1;
  }
  history.pushState(slideNumber, null, "ChinaSites" + slideNumber + ".html");
  goToNewSlide();
  return false;
}

function previousSlide() {
  if (slideNumber == 1) {
    slideNumber = 5;
  } else {
    slideNumber -= 1;
  }
  history.pushState(slideNumber, null, "ChinaSites" + slideNumber + ".html");
  goToNewSlide();
  return false;
}
```

goToNewSlide()函数与本例第1版（参见12.1.3节）中的一样，仍然用XMLHttpRequest对象异步地为下一张幻灯片获取数据。

图13-8显示了新的URL管理机制的运行效果 。

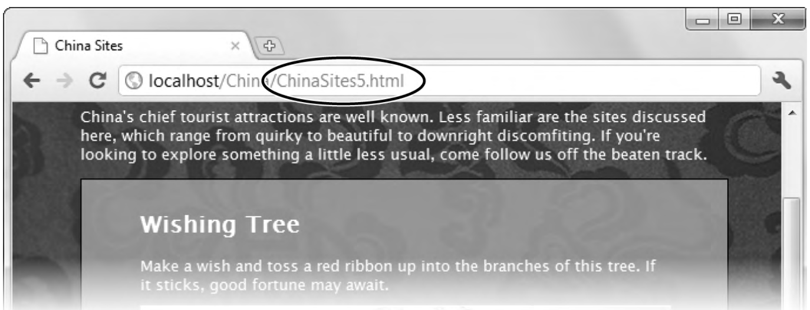


图13-8：访客单击浏览幻灯片时，URL会随之变化，与之匹配。这时的URL既清晰，又符合直觉，因为它对应着每一张幻灯片

在使用pushState()方法的同时，还应该考虑onPopState事件，它们可以说是“天生一对”。这么说是因为pushState()方法会向浏览器的历史记录中存入一个新状态，而onPopState事件则意味着用户返回了某个状态，这也就为处理状态变化提供了机会。

说了半天，我们还是举个例子。假设访客会看完所有幻灯片，随着他的点击，地址栏的URL会从ChinaSites.html变成ChinaSites1.html、ChinaSites2.html、ChinaSites3.html……即使页面并没有真的改变，但这些URL也会保存到浏览器的历史记录中。如果用户返回上一张幻灯片（比如，从ChinaSites3.html返回ChinaSites2.html），就会触发onPopState事件。而这个事件的事件对象中包含着原来通过pushState()方法保存的状态信息。你的工作就是根据这些信息，恢复到网页的适当版本。对于眼下的例子来说，也就是加载正确的幻灯片：

```
window.onpopstate = function(e) {
  if (e.state != null) {
    //这个状态的幻灯片编号是多少?
    //（当然也可以从URL中取得这个编号值
    //但使用location.href属性会比较麻烦）
    slideNumber = e.state;

    //从Web服务器请求相应的幻灯片
    goToNewSlide();
  }
};
```

以上代码首先检测是否存在state对象，如果存在再继续下面的工作。之所以这样做，是因为有些浏览器（包括Chrome）会在初次加载页面时就触发onPopstate事件，即便当时并未调用pushState()方法。

注意 history对象中还有一个方法，但用得不多，那就是replaceState()。可以使用replaceState()修改与当前页面相关的状态信息，且不会向历史记录添加任何记录。

为达到URL的要求而创建额外的页面

会话历史遵循最初的Web哲学：每一段内容都由一个唯一且持久的URL标识。这就意味着每个URL都必须让访客找到他原来看到过的内容，不容易啊。比如，要是有人请求了ChinaSites3.html，你就得从ChinaSites.html中取得公共内容，然后再从ChinaSites3_slide.html中取得幻灯片的内容，把它们显示到一块。

如果你是经验丰富的程序员，那可以编写一段服务器端代码，解释Web请求，动态完成这一组合过程。但如果你没有那么多经验，则要选择其他手段。

最简单的办法就是给每个URL创建一个网页文件。换句话说，必须逐个创建ChinaSites1.html、ChinaSites2.html、ChinaSites3.html，等等。当然，没有必要把同一个幻灯片的内容放在两个文件里（比如在ChinaSites3.html和ChinaSites3_slide.html中保存相同的幻灯片），因为这些样维护起来太麻烦。好在有两个简单的办法可以帮上忙。

- **服务器端包含**。如果你的Web服务器支持服务器端包含技术（大多数都支持），可以使用以下指令：

  ```
  <!--#include file="footer.html" -->
  ```

  虽然这看起来像条注释，但这个指令是在告诉Web服务器打开指定的文件，并把其内容插入它所在的位置。这样，就可以把公共内容和幻灯片组合在一个特定的页面中。事实上，对应于每张幻灯片的文件（ChinaSites1.html、ChinaSites2.html等）都仅需几行这样的标记，就可以创建出一个页面容器来。
- **使用网页设计工具中的模板**。有些网页设计工具，比如Adobe Dreamweaver，支持Web模板功能，基于模板可以创建任意多个页面。只要创建出包含公共内容和样式的模板，就可以通过重用它来迅速简单地创建出针对所有幻灯片的页面。

13.3.4 浏览器对会话历史的支持情况

浏览器对会话历史功能的支持和对Web Worker的支持情况差不多。IE10之前的版本基本不支持（参见表13-2）。

表13-2 浏览器对会话历史的支持情况

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android
最低版本	10	4	8	5	11.5	5	4.2

对于不支持会话历史的浏览器，有两种处理方式。如果你置之不理，那就不会出现那些符合直觉的URL。如果用IE查看前面的例子，结果就是这样。无论你加载的是哪个幻灯片，URL始终都是ChinaSites.html。Flickr在显示自己的照片时也采用类似手段（请用旧版IE查看这个链接：http://tinyurl.com/6hnvanw）。

另外，也可以在浏览器不支持会话历史时，触发页面刷新来加载新内容。如果提供容易识别的、有意义的URL比动态加载内容还重要，这样就做是合情合理的。但是，这种方案也需要更多工作来实现。

有一种简单的方法，就是增强导航逻辑，以便可以在必要的时候重定向到新页面。在ChinaSites.html页面中，这包括增强goToNextSlide()函数，代码如下：

```
function goToNewSlide() {
  if (window.history) {
    // 浏览器支持会话历史
    if (req != null) {
      req.open("GET", "ChinaSites" + slideNumber + "_slide" + ".html", true);
      req.onreadystatechange = newSlideReceived;
      req.send();
    }
    else {
      // 有一个问题。忽略它
    }
  }
  else {
    // 浏览器不支持会话历史，所以重定向到新页面
    window.location = "ChinaSites" + slideNumber + ".html"
  }
}
```

这段代码用window.history属性来检查是否支持会话历史。如果支持，就下载一小段所需的幻灯片数据并将其加载到已存在的页面中；如果不支持，就放弃这种优雅的方案，改用保守的重定向到新页面方式。

Part 4 第四部分

附　　录

本部分内容

附录A

CSS基础

毫不夸张地说，没有CSS（也就是“层叠样式表”）标准，就没有现代的Web设计。即便是格式极为丰富、构图极为复杂的网页，也可以通过CSS把格式化工作转移到一个外部文件——样式表里。这样一来，网页标记就可以非常清爽、清晰、易读。

要想最大限度地发挥HTML5（以及本书）的功用，必须得熟悉CSS标准。如果你本来就是CSS专家，这个附录可以不看，只关注本书其他内容即可；另外要特别注意第6章和第7章，其中介绍了CSS3新增的许多功能。但如果你的CSS技术没有那么好，那本附录可以帮你补充一些基础知识，以便理解本书其他内容。

注意 本附录只是概要地介绍了CSS，并没有面面俱到。如果看完本附录，你还觉得有些问题没搞明白，建议找一本专门讲CSS的书，比如David Sawyer McFarland的*CSS3: The Missing Manual*。

A.1　在网页中添加样式

有三种在网页中使用样式的方式。

第一种是直接把样式信息嵌入到元素里，这就要用到style属性。下面这个例子展示了如何修改标题文本的颜色：

```
<h1 style="color: green">Inline Styles are Sloppy Styles</h1>
```

这种方法非常方便，但会令标记杂乱无章，要一个一个地在每一行添加样式。

第二种是把全部样式嵌入到<style>元素里，而这个<style>元素要放在页面的<head>部分：

```
<head>
  <title>Embedded Style Sheet Test</title>
  <style>
   ...
  </style>
</head>
```

这样写代码能够把样式与标记分开，但最终它们还是在一个文件里。这种方式适合一次性的

样式（也就是不想在其他页面中重用的样式），也适合简单的测试和示例，就像本书中的示例页面一样。但是，对于真正的专业站点而言，这种做法不值得提倡，因为页面因此会变得臃肿不堪。

第三种方式是使用`<link>`元素在`<head>`部分链接外部样式表文件。下面这个例子告诉浏览器应用名为SampleStyles.css的外部样式表中的样式：

```
<head>
  <title>External Style Sheet Test</title>
  <link rel="stylesheet" href="SampleStyles.css">
</head>
```

这种方式最常用，效果也最好。而且，通过这种方式还能在其他页面间重用样式。如果你愿意，还可以把样式分割到多个样式表，然后在任意HTML页面中链接任意多个你需要的那些样式表。

注意 现代Web开发建立在一个简单的原理基础上。HTML标记用于把页面结构化为逻辑区块（段落、标题、列表、图片、链接等），而CSS样式表用于格式化（通过指定字体、颜色、边框、背景和布局）。遵守这个规则，网页就容易编辑。如果要修改整个网页的格式和布局，只要修改它所链接的样式表即可。（要了解样式表真正的魔力，建议看看“CSS禅意花园”，地址为www.csszengarden.com；其中，相同的网站通过不同的样式表，呈现出了200多种不同的面貌。）

A.2 样式表解析

一个样式表就是一个文本文件，在Web服务器上通常与HTML页面放在一起。样式表中包含若干样式规则，规则的先后顺序不重要。

每条样式规则会为一或多个HTML元素指定一或多个格式化信息。下面是简单的样式规则的结构：

```
selector {
  property: value;
  property: value;
}
```

以下是样式规则各个部分的说明。

- `selector`（选择符）：表示要格式化什么内容。浏览器会在整个页面中查找选择符想要匹配的元素。编写选择符的方式也不止一种，但最简单的就是直接给出你想要为其应用样式的元素名（如下所示）。例如，可以编写一个选择符，选出页面中的所有一级标题。
- `property`（属性）：表示要应用什么样式。属性就是颜色、字体、对齐方式，等等，一条样式规则里可以设置任意多个属性——这个例子里是两个属性。
- `value`（值）：表示给样式设置什么样的值。例如，如果属性是颜色，那么值可以是浅蓝色或淡绿色。

好了，下面看一个真正的样式规则：

```
h1 {
  text-align: center;
  color: green;
}
```

把这条规则复制粘贴到样式表里（比如，保存在SampleStyles.css中），然后找一个简单的网页（至少要包含一个<h1>元素），添加一个<link>元素引用该样式表。最后，在浏览器中打开这个网页，你就会发现<h1>元素不再是其默认的格式，而是会变成绿色并居中。

A.2.1　CSS属性

前面的例子介绍了两个格式化属性：text-align（设置文本在水平方向上如何对齐）和color（设置文本的颜色）。

除此之外，还有很多很多可以使用的格式化属性。表A-1列出了其中一些最常用的。事实上，这个表中列出了本书示例中出现过的几乎所有样式属性（不包括第6章和第7章中介绍的更新的CSS3属性）。

表A-1　按类查看的常用样式属性

	属　　性
颜色	color background-color
空距	margin padding margin-left、margin-right、margin-top、margin-bottom padding-left、padding-right、padding-top、padding-bottom
边框	border-width border-style border-color border（一次性设置宽度、样式和颜色）
文本对齐	text-align text-indent word-spacing letter-spacing line-height white-space
字体	font-family font-size font-weight font-style font-variant text-decoration @font-face（关于使用自定义字体，请参考6.4节）

（续）

	属　　性
尺寸	width height
布局	position left、right float、clear
图片	background-image background-repeat background-position

提示　如果你手边没有其他样式表参考书，可以访问 http://www.htmldog.com/reference/cssproperties，里面有表里列出的（以及更多的）属性。你可以找到每个属性的信息，包括简介和允许哪些值。

A.2.2　使用类格式化正确的元素

前面的样式规则格式化的是所有文档中的<h1>标题。但在比较复杂的文档中，则需要指定具体的元素，为它们应用不同的样式。

为此，需要使用class属性为这些元素起个名字。下面是一个设置类名为ArticleTitle的例子：

```
<h1 class="ArticleTitle">HTML5 is Winning</h1>
```

好了，现在可以只为这个标题写一条样式规则了。关键在于选择符要以一个句点开头，然后才是类名，像这样：

```
.ArticleTitle {
  font-family: Garamond, serif;
  font-size: 40px;
}
```

这样，表示文章标题的<h1>元素就放大到了40像素高。

可以给任意多个元素指定相同的类属性。事实上，这正是发明类属性的用意所在。几乎所有样式表里都可以看到类选择符规则，这些类选择符把网页标记有效地分成了可以承载样式的单位。

最后，有必要提一下：可以组合使用元素类型和类名。比如：

```
h1.ArticleTitle {
  font-size: 40px;
}
```

这个选择符匹配了所有类为ArticleTitle的<h1>元素。有时候，这样写只是为了清晰而已。比如，你可能想用这种规则说明只为<h1>元素添加ArticleTitle类，而其他元素都不会有这个类。但大多数情况下，Web设计人员只会给出一个类名，不会限定任何元素。

注意 不同的选择符可以层叠。如果有多个选择符都选择了同一个元素，那么这些选择符背后的样式都会起作用，但首先会应用最通用的样式。比如，有一条规则适用于所有标题，另一条规则适应于类名为ArticleTitle的元素，那么会先应用针对所有标题的样式，然后应用类规则。结果，就是类规则会覆盖应用给所有标题的规则。如果两条规则的针对性一样，那么会应用出现在样式表后面的那个规则的样式。

A.2.3 样式表注释

在复杂的样式表中，有时候需要写一些说明来提醒自己（或其他人）为什么要写某条规则，以及该规则的目的是什么。与HTML一样，在CSS中也可以写注释，浏览器同样会忽略这些注释。不过，CSS注释与HTML注释不一样。CSS注释以/*字符开头，以*/字符结尾。下面这条注释虽然没有多大意义，但它示范了注释的语法：

```
/*这是页面中文章的标题*/
.ArticleTitle {
  font-size: 40px;
}
```

A.3 高级一点的样式表

稍后我们会介绍一个实用的样式表。不过，首先还是来看几个编写样式时常用的高级技巧。

A.3.1 用<div>元素为网页

在编写样式表时，我们经常要用<div>元素来包装内容：

```
<div>
  <p>Here are two paragraphs of content.</p>
  <p>In a div container.</p>
</div>
```

就其本身而言，<div>什么也不做。但是有了它，就可以基于类来应用一些样式。下面是一些可能的做法。

- **继承的值**。有些CSS属性是可以继承的，也就是在一个元素上设置的值可以自动应用给该元素内部的所有元素。比如字体属性，在<div>元素上设置了该属性后，这个元素内部的所有属性都会应用同样的字体样式（除非你在具体的元素上覆盖这些规则）。
- **盒模型**。一个<div>元素就是一个自然的容器。可以给它添加边框、空距和不同的背景颜色（或背景图片），从而让它在页面中更加显眼。
- **分栏**。专业的网站通常会把内容分成两栏或三栏。实现分栏的一种方式就是把每一个栏的内容包装在一个<div>元素中，然后再使用CSS定位属性将它们放到适当的位置上。

提示 既然HTML5已经引入了一套新的语义元素，<div>元素的地位就不那么重要了。如果可以把<div>元素替换成其他更有语义的元素（如<header>或<figure>），只管替换好了。但在没有适当元素的情况下，<div>元素仍然是个不错的选择。第2章详细介绍过HTML5中的所有语义元素。

<div>元素还有一个小兄弟，叫做<span>。与<div>类似，<span>元素也没有内置样式。但不同的是，<div>是块级元素，用于分隔段落或整块内容；而<span>则是行内元素，用于在块级元素中包装少量内容。比如，可以用<span>元素在段落中包装几个单词，然后给它们应用特殊的样式。

注意 CSS鼓励优秀设计。怎么鼓励的？如果你想有效地使用CSS，必须事先规划好网页结构。这种对CSS的需求就会鼓励人们认真思考如何组织自己的内容，即便是临时的页面设计人员也不例外。

A.3.2 多个选择符

有时候，你可能需要定义一些样式，把它们应用给多个元素或多个类。此时，你可以在选择符之间加上逗号。

比如，下面这两级标题，分别有不同的字体大小，但有相同的字体：

```
h1 {
  font-family: Impact, Charcoal, sans-serif;
  font-size: 40px;
}

h2 {
  font-family: Impact, Charcoal, sans-serif;
  font-size: 20px;
}
```

你可以把font-family属性单独放到一条规则里，把它应用给两级标题，比如：

```
h1, h2 {
  font-family: Impact, Charcoal, sans-serif;
}

h1 {
  font-size: 40px;
}

h2 {
  font-size: 20px;
}
```

关键在于，这样写样式不是优秀设计所必须的。通常，重复设置某个属性反倒可以增加将来修改样式的灵活性。假如共享的属性太多，那么很难做到修改一个元素类型或类，而不影响其他元素。

A.3.3　上下文选择符

上下文选择符用于匹配位于另一个元素内部的元素。看一下例子：

```
.Content h2 {
  color: #24486C;
  font-size: medium;
}
```

这个选择符先会查找带有Content类的元素，然后再在该元素中查找<h2>元素，找到之后为它们应用不同的文本颜色和字体大小。下面这段标记展示了会应用该样式规则的元素：

```
<div class="Content">
  ...
  <h2>Mayan Doomsday</h2>
  ...
</div>
```

在第一个例子中，第一个选择符是一个类选择符，第二个选择符（即上下文选择符）是一个元素类型选择符。不过，你可以根据自己的需要进一步修改，比如下面这个例子：

```
.Content .LeadIn {
  font-variant: small-caps;
}
```

这个选择符会查询类为LeadIn的元素，但它必须被包含在类为Content的元素中。比如，它匹配下面的元素：

```
<div class="Content">
  <p><span class="LeadIn">Right now</span>, you're probably feeling pretty
  good.  After all, life in the developed world is comfortable ...</p>
  ...
</div>
```

熟悉了上下文选择符的用法之后，你会发现这是一种非常直接的方式，也非常有用。

A.3.4　ID选择符

类选择符还有一个近亲，叫做ID选择符。与类选择符相似，ID选择符可以让你只为选定的元素应用样式。而且，同样与类选择符相似，使用ID选择符也可以挑一个描述性的名字。只不过，不能再使用句点，而要使用井号（#），如下所示：

```
#Menu {
  border-width: 2px;
  border-style: solid;
}
```

与类规则相似，除非你在HTML中指定ID，否则浏览器不会应用相应的样式。不过，这次不是使用class属性，而是要使用id属性。比如，以下这个<div>元素就可以应用前面的#Menu样式。

```
<div id="Menu">...</div>
```

此时，可能有人会问——为什么要用ID选择符，难道ID选择符与类选择符有什么区别吗？的确是有区别：一个ID只能指定给页面中的一个元素。以刚才的标记为例，页面中只能有一个`<div>`元素（以及其他元素）可以带有`Menu`这个ID。但类属性则没有这个限制，同一个类名可以随便应用给任意多个元素。

这就意味着，ID选择符非常适合用来为那些一个页面中唯一的、不会重复的元素应用样式。而这也体现了使用ID选择符的一个优势，那就是清晰地表明某个元素特别重要。比如，页面中可能有一个ID选择符叫`Menu`或`NavigationBar`，那么设计师就知道页面中只有一个菜单或导航条。当然，并不是非要使用ID选择符不可。有些Web设计师会在任何情况下都使用类选择符，无论标识的区块是不是唯一。这只能说是萝卜白菜各有所爱。

注意 ID属性在JavaScript中同样扮演着重要角色，它可以让开发人员取得页面中的特殊元素，然后在代码中操作该元素。本书中的示例只要是有为JavaScript代码准备好了ID的，就会使用ID选择符来应用样式规则，这样就避免了同时为元素设置ID和类属性。否则，示例中就会使用类选择符为相应的元素应用样式，无论该元素在页面中是否唯一。

A.3.5 伪类选择符

目前，我们看到的选择符都是很直观的。它们一般都只考虑某个显而易见的特点，比如元素类型、类名或者ID属性值。伪类选择符就没有那么好理解，因为还要考虑其他方面。所谓其他方面，指的是那些标记中并不存在，或者要根据用户操作来确定的信息。

CSS过去很长时间只支持几个伪类，其中又有大部分专门为链接而设计。比如，`:link`伪类用于为新的、未访问过的链接应用样式，`:visited`伪类用于为访问过的链接应用样式，`:hover`伪类用于为用户鼠标悬停状态下的链接应用样式，而`:active`伪类用于为鼠标点击且尚未抬起状态下的链接应用样式。你也看到了，伪类始终以一个冒号（:）开头。

下面的样式规则使用伪类创建故意让人迷惑的页面，也就是说访问过的链接是蓝色，而未访问过的链接是红色：

```
a:link {
  color: red;
}
a:visited {
  color: blue;
}
```

伪类也可以与类名一起用：

```
.BackwardLink:link {
  color: red;
}
.BackwardLink:visited {
  color: blue;
}
```

那么，为了应用这个新样式的链接元素可能如下所示：

```
<a class="BackwardLink" href="...">...</a>
```

伪类并不是只能用来为链接添加样式。比如，还可以用:hover伪类来创建动画效果和好玩的按钮。当然，这要用到6.3节介绍的CSS的过渡功能。

注意 CSS3还增加很多更高级的伪类，涉及很多其他细节，比如元素相对于其他元素的位置，或者表单中输入控制的状态。本书不会介绍这些伪类，但有兴趣的话，建议读者看看Smashing Magazine上的这篇文章：http://tinyurl.com/pc-css3。

A.3.6 属性选择符

属性选择符是CSS3提供的功能，可以选择属性为特定值的特定类型的元素。以下面这个样式规则为例，它只适用于文本模式：

```
input[type="text"] {
  background-color:silver;
}
```

首先，这个选择符会取得所有<input>元素。然后，它会进一步筛选出type属性等于"text"的那些<input>元素，只对这些元素应用样式。对于下面的标记而言，只有第一个<input>元素会带有银色背景，第二个元素则不会：

```
<label for="name">Name:</label><input id="name" type="text"><br>
<input type="submit" value="OK">
```

严格来讲，不必在第一个<input>元素中指定type="text"，因为这是它的默认值。不指定的话，前面的属性选择符照样有效，因为它只关注属性的当前值，而不关注这个值是不是在标记中指定的。

类似地，也可以另外创建一条规则，只应用给文本框的标题，忽略其他标签：

```
label[for="name"] {
  width: 200px;
}
```

注意 对属性选择符，也可以发挥一点想象力。比如，可以同时匹配几个属性值，或只匹配某个属性值的一部分。这些技术非常有创意，但却会给普通的样式表平添很多复杂性。要了解CSS3标准中的选择符，请参考http://tinyurl.com/s-css3。

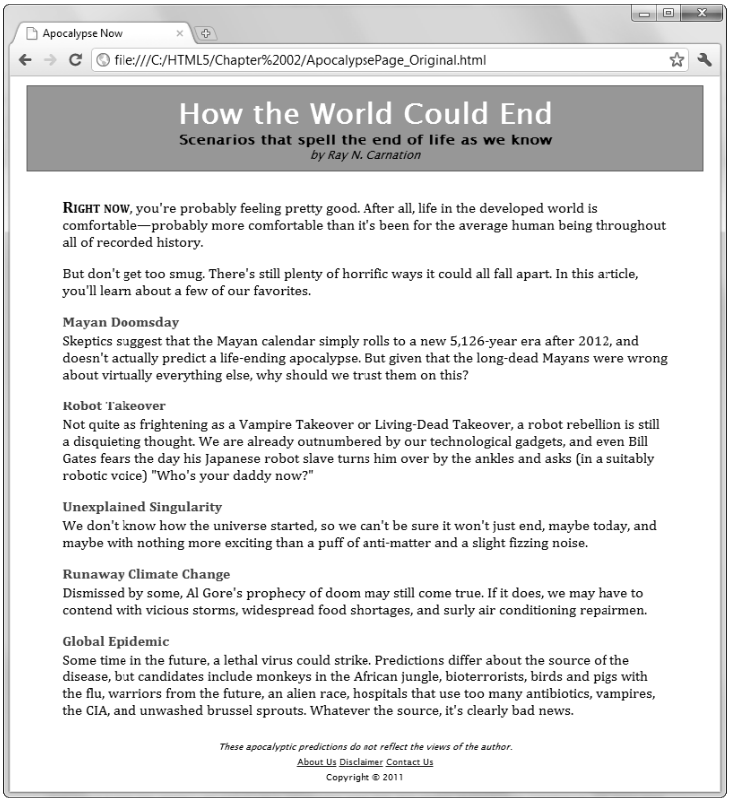
A.4 从头写一个样式表

第2章在学习HTML5语义元素时，我们使用了一外观漂亮的示例页面，文件名是

ApocalypsePage_Original.html（参见图A-1）。这个页面链接到了一个外部样式表文件，叫ApocalypsePage_Original.css：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Apocalypse Now</title>
  <link rel="stylesheet" href="ApocalypsePage_Original.css">
</head>
...
```

这个样式表比较简单明了，总共也就50行左右。本节我们就来分析这个样式表中的规则。



图A-1：这个页面的样式表很简单，但它却体现了本书始终遵循的基本组织原则

这个样式表一开始是一个针对<body>元素的选择符，<body>是整个网页的根元素。这个元素上最适合设置那些可以继承的值，默认情况下，应用给<body>元素的可继承值也会应用给文档中的其他元素。可继承值包括外边距、内边距、背景颜色、字体和宽度等：

```
body {
  font-family: "Lucida Sans Unicode", "Lucida Grande", Geneva, sans-serif;
  max-width: 800px;
}
```

设置font-family这个CSS属性，要遵循两条规则。首先，要使用Web安全字体，也就是那些已知的所有能上网的计算机中都会安装的字体（参见 http://tinyurl.com/ws-fonts）。其次，列出字体时，要按照先特殊后一般的原则，先给出你想要的某种字体，然后是保险一些的后备字体，最后以serif或sans-serif（这两种所有浏览器都能理解的）字体结尾。如果你想特别想标新立异，希望用户从你的Web服务器上下载字体，可以参考6.4节有关CSS3网路字体的内容。

针对页面主体的样式规则设置了最大宽度，不超过800像素。这条规则可以避免文本行过长，无法阅读的问题，特别当浏览器窗口非常宽的情况下。处理文本行过宽的问题还有其他方法，比如把文本分成几栏（参见6.4.6节），使用CSS媒体查询（参见7.2节），或者在布局上增加一个边栏以利用多余空间。不过，设置为固定的800像素宽度还是比较常用的手段，尽管不是很刺激。

接下来，在样式表中写一个针对类的规则，用于为页面顶部的标题区添加样式：

```
.Header {
  background-color: #7695FE;
  border: thin #336699 solid;
  padding: 10px;
  margin: 10px;
  text-align: center;
}
```

注意　在这个示例中，页面头部用一个类名为Header的<div>元素。而第2章则探讨了为什么最好将它替换成HTML5的<header>元素。

这条规则里使用了很多属性。其中，background-color属性可以接受任何CSS颜色值，比如颜色名（可用颜色相对少一些）、HTML颜色代码（如上所示），或者rgb()函数（可以指定红、绿、蓝分量的多少）。本书中的示例用到了所有这三种手段，在简单的例子中用的是颜色值，在接近实际的例子中用的是颜色代码和rgb()函数。

顺便提一下，所有HTML颜色代码都可以用rgb()函数来表示，反之亦然。比如，前面的颜色代码如果用rgb()函数写，就是这样：

```
background-color: rgb(118,149,254);
```

提示　要知道自己想要的颜色的RGB值，可以找个在线颜色拾取器，或者使用自己常用的平面或插画绘图软件。

头部规则也为四周应用了一条细边框。这里使用的是多合一的border属性，分别指定了边框粗细、边框颜色和边框样式（边框样式可以设置为solid、dashed、dotted、double、groove、ridge、inset、outset等），一个属性搞定。

设置了背景颜色和边框之后，接着又设置了10像素的内边距（即边框与内容之间的距离），10像素的外边距（即边框与周围元素之间的距离）。最后，将头部中的文本居中。

接下来，再写一个上下文选择符，控制头部中元素的格式。首先是头部中的<h1>元素：

```
.Header h1 {
  margin: 0px;
  color: white;
  font-size: xx-large;
}
```

提示 设置字体大小时，可以使用关键字（如这里的xx-large等）。另外，如果你希望对字体控制得更精确，可以使用像素或em单位。

再写两条类规则，分别针对类名Teaser和Byline，它们也在<header>里：

```
.Header .Teaser {
  margin: 0px;
  font-weight: bold;
}

.Header .Byline {
  font-style: italic;
  font-size: small;
  margin: 0px;
}
```

这两条规则能起作用，是因为头部包含两个<span>元素。一个<span>元素有类名Teaser，包含副标题；另一个<span>元素的类名是Byline，包含作者信息，下面是相关代码：

```
<div class="Header">
  <h1>How the World Could End</h1>
  <p class="Teaser">Scenarios that spell the end of life as we knowit</p>
  <p class="Byline">by Ray N. Carnation</p>
</div>
```

下面为类名为Content的<div>写一条规则，这个元素中包含着页面的主要内容。规则中的样式涉及字体、内边距、行高，等等：

```
.Content {
  font-size: medium;
  font-family: Cambria, Cochin, Georgia, "Times New Roman", Times, serif;
  padding-top: 20px;
  padding-right: 50px;
  padding-bottom: 5px;
  padding-left: 50px;
  line-height: 120%;
}
```

与头部规则为四边设置相同的内边距不同，这条规则为页面内容的各边设置了不同的内边距。具体来说，上内边距加大了，而左、右内边距最大。为各边设置不同边距的一种做法就像这里这样，使用padding-top、padding-right等属性。另一种做法是给padding属性提供一串特定顺序（上、右、下、左）的值。下面这一条属性就可以取代上面的三条：

```
padding: 20px 50px 5px 50px;
```

一般来说，这种方式可以为元素各边设置内边距，而针对各边的扩展属性则可以只设置某一边的内边距。当然，哪种方式都可以，要看你自己的喜好。

最后一个line-height属性设置的是相邻两个文本行之间的距离。这里的120%给出了额外的空间，让人阅读起来更容易。

接着要写三个上下文选择符，分别为内容中的三个元素应用样式。第一条规则为类名为LeadIn的<span>元素应用样式，主要是把其中的前两个单词加大，变成粗体和小型大写字母：

```
.Content .LeadIn {
  font-weight: bold;
  font-size: large;
  font-variant: small-caps;
}
```

后面两条规则修改<h2>和<p>元素：

```
.Content h2 {
  color: #24486C;
  margin-bottom: 2px;
  font-size: medium;
}

.Content p {
  margin-top: 0px;
}
```

看到了吧，虽然样式表越来越长，但它却没有变复杂。所有样式规则都在简单重复基本的东西（类选择符和上下文选择符），而这些基本的东西就可以为文档的每个部分添加样式。

最后，我们再写几条为页面末尾的脚部添加样式的规则。都现在了，你自己能看明白，我就不解释了：

```
.Footer {
  text-align: center;
  font-size: x-small;
}

.Footer .Disclaimer {
  font-style: italic;
}

.Footer p {
  margin: 3px;
}
```

这样我们的ApocalypsePage_Original.css样式表就完成了。如果你没跟着写，也可以到本书配套站点（http://prosetech.com/html5）下载，然后在已有样式基础上改一改，看看结果会怎么样。或者，再去看看本书第2章，其中利用HTML5的语义元素重写了这个页面，样式表也相应作了调整。

附录 B

JavaScript：页面的大脑

过去曾经有一段时间，网上只有标记。那时候，页面里只包含文本和HTML标签，除此之外就没有什么别的了。真正先进的网站会用到服务器端脚本，即在服务器上先动态生成HTML标记，然后再发给浏览器；不过，也仅此而已。

今天，随便打开一个页面，都可能会看到大量JavaScript代码。这些JavaScript代码驱动着网站的一切内容，从重要功能到给页面添加好看的装饰。自动完成的文本框、弹出式菜单、幻灯片展示、实时绘图，以及在线电邮系统，这些只是高明的开发人员使用JavaScript所能实现的无数功能中的一小部分。实际上，没有JavaScript的Web是无法想象的。虽然HTML依旧是Web的核心语言，但JavaScript如今俨然成了大多数高级页面的中枢神经系统。

本附录是一个极度浓缩的JavaScript教程。换句话说，这个附录不会面面俱到地介绍JavaScript，也不会让一个没写过一行代码的人马上就能上手写程序。不过，要是你有一些编程语言的基础知识——比如学过Visual Basic，懂得Pascal的基础知识，或者有C语言的经验，那么本附录可以帮你转换到JavaScript的语境之下。对变量、循环、条件逻辑等常见的语言要素，我们都会涉及。另外，对于本书JavaScript示例中用到的基本语言要素，我们也会介绍到。

提示 如果你没有把握能轻松上手，可以再看看David Sawyer McFarland的*JavaScript & jQuery: The Missing Manual*。该书介绍了jQuery这个流行的JavaScript工具包。另外，Mozilla详尽的JavaScript参考指南也值得学习：http://developer.mozilla.org/JavaScript。

B.1 在网页中使用JavaScript

在接触JavaScript代码之前，需要知道把它们放在网页的什么地方。当然是以`<script>`元素开头啦。接下来的几节会展示如何将一个包含临时应急的JavaScript代码的网页，组织成结构合理、适合在线部署的页面。

B.1.1 在标记中嵌入脚本

使用`<script>`元素的最简单方式，就是把它放在HTML标记中的某个地方，比如：

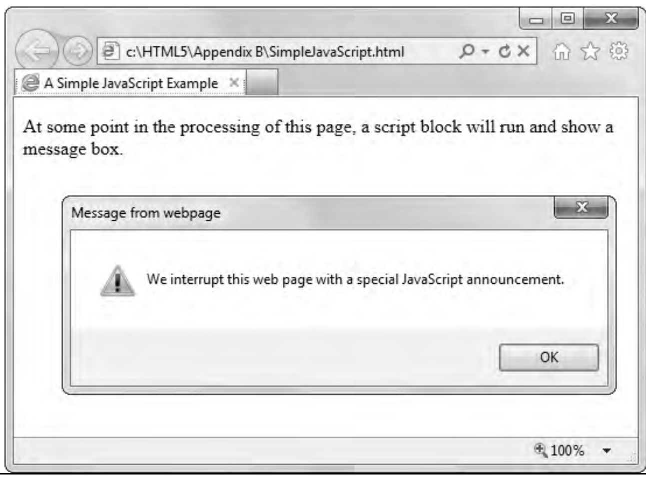
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>A Simple JavaScript Example</title>
</head>

<body>
  <p>At some point in the processing of this page, a script block
will run and show a message box.</p>

  <script>
alert("We interrupt this web page with a special JavaScript announcement.");
  </script>

  <p>If you get here, you've already seen it.</p>
</body>
</html>
```

这里的脚本块中只包含一行代码。不过，要在其中加入一系列操作也并非难事。就这个例子而言，这行代码会触发JavaScript内置的alert()函数。这个函数接收一段文本，然后通过一个消息框显示出来（参见图B-1）。要继续浏览网页，用户必须单击OK按钮。



图B-1：浏览器遇到JavaScript代码时，不会立即运行它。事实上，浏览器会暂时停止页面处理。此时，除非用户按下OK按钮关闭消息框，否则代码会一直处于等待状态。按下OK按钮，代码就可以执行到脚本块的末尾，而浏览器也可以继续处理其他标记

注意 这个例子中体现了JavaScript代码的一个最重要的约定，这个约定本书始终在遵守，而且规范的网站脚本中也都会注意，那就是分号。在JavaScript中，分号表示每条语言的结束。严格地讲，分号有时候并不是必需的（除非你把多条语句写在一行上）。不过，始终给每条语言末尾加上分号还是良好的习惯。

如果你希望立即运行JavaScript（就像这个例子这样），可以把<script>区块放在<body>区块的最后，正好位于</body>标签之前的地方。这样，脚本就会在浏览器刚好处理完所有标记之后运行。

处理Internet Explorer的“怪癖”

如果是在 Firefox 或 Chrome 中运行这个弹出警告框的例子，那你不会发现任何问题。如果是在 Interent Explorer 中运行，结果可能就大不一样。你可能会看到页面顶部或底部出现一个黄色的警告条（取决于 IE 的版本）。如果不单击那个黄条上的 Allow Blocked Content（允许阻塞的内容），那 JavaScript 代码就不会执行。

IE 的安全警告这不是要把用户吓跑吗？别担心，这个警告只是 IE 在运行本地硬盘上的网页时的一个常见“怪癖”。如果你是在网上打开同一个页面，IE 不会给出这个多此一举的警告。

换句话说，至少在本地测试网页时，这个安全警告会很讨厌。想想，每次都明确地告诉浏览器允许页面运行 JavaScript，实在太麻烦了。为了解决这个问题，可以让 IE 以为你是从 Web 服务器下载页面。怎么做呢？就是在页面头部加一条所谓的“mark of the Web”注释：

```
<head>
<meta charset="utf-8">
<!-- saved from url=(0014)about:internet -->
  ...
</head>
```

IE 看到这个注释后，就会像页面来自 Web 服务器一样处理它。也就是说，不会再显示那个安全警告，义无反顾地运行 JavaScript 代码。对于其他浏览器来说，这条注释跟普通的 HTML 注释没有分别，因此会被浏览器忽略。

B.1.2　函数

前面例子有一个问题，那就是把代码和标记混在了一起。为了让代码井井有条，应将代码要完成的每个“任务”包装到一个函数里。函数，就是一个可执行的代码单位，在需要的时候可以调用它。

创建函数时，要记得给函数起个描述性的名字，比如这个叫showMessage()的函数：

```
function showMessage() {
  //这里是函数的代码……
}
```

函数体，就像函数的五脏六腑，包括了从起始花括号（{）到结束花括号（}）之间的所有内容。在这两个界定符内，可以写任意多的代码。这个showMessage()函数只有一行，就是“//这里是函数代码……”这行注释（JavaScript的单行注释以两条斜杠开头。浏览器会忽略注释，写注释是为了提醒自己或他人与代码相关的重要细节）。

要给函数添加代码，只要把所有语句添加到花括号中即可：

```
function showMessage() {
  alert("We interrupt this web page with a special JavaScript announcement.");
}
```

当然，所有代码都必须放在一个<script>块里。而在页面中放置JavaScript函数的最佳位置就是<head>区块。通过把代码转移到一个专门的地方，可以让页面内容分门别类，井井有条。

以下是对前面示例代码进行修改后的结果，这次使用了函数：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>A Simple JavaScript Example</title>
  <script>
function showMessage() {
  alert("We interrupt this web page with a special JavaScript announcement.");
}
  </script>
</head>
...
```

函数，如果只是定义它，其实什么也不会做。要触发函数执行，必须有代码调用它。

调用函数很简单，我们不是已经调用过alert()函数了嘛。对，就是写出函数名，然后再加上一对圆括号。在这对圆括号里，可以向函数传入数据。或者，如果函数本身不接收数据，比如这里的showMessage()，那也可以什么也不传：

```
...
<body>
  <p>At some point in the processing of this page, a script block
will run and show a message box.</p>
 <script>
showMessage();
 </script>
  <p>If you get here, you've already seen it.</p>
</body>
</html>
```

这个例子中，函数和调用函数的代码在不同的<script>块里。不是必须要这样的，这里这样用是有意强调这两块的分离。

乍一看，添加一个函数似乎让这个例子比以前复杂了。但这其实是设计上一个很大的进步，且听我慢慢说。

- **代码已经脱离了标记**。只要一行代码就可以调用函数。不过，真正的函数会包含很多代码，而真正的网页也会包含大量函数。所以，肯定要把代码与标记分开。
- **可以重用代码**。把代码以函数形式包装起来之后，任何时候都可以调用它，也不用管是在代码的什么地方。对于这个简单的例子，这一点似乎不明显，而对于像第8章那样较为复杂的例子，这个优点就非常重要了。
- **只差一点就可以把脚本转移到外部文件中了**。把代码从标记中移出来可以为建立单独的脚本文件作好准备，下一节就会介绍到，引用外部代码文件更容易组织和管理。
- **可以使用事件了**。什么是事件？事件就是在某件事发生时，可以告诉页面执行某个函数。在事件驱动的页面中，有很多代码是靠事件来触发执行的（而不是加载后立即执行）。事件与函数相辅相成，具体细节请参考B.1.4节。

注意　一个网页可以包含任意多个<script>块。

B.1.3　把代码转移到脚本文件中

把JavaScript代码集中到一块，特别是集中到一个函数里，是让代码有序的第一步。第二步就是把脚本代码放到一个完全独立的文件中。这样一来，网页的体积就会变小，同时函数还可以在更多网页中得到最大限度的重用。事实上，把脚本代码放在外部文件里，与把CSS样式规则放在外部文件里异曲同工。都能简化页面，同时获得更大的灵活性。

注意　真正设计良好的网页都会把JavaScript代码放在一或多个文件里。当然也有例外，那就是不会重用的、数量很少的代码，或者一次性示例的代码，仍然可以放在网页里。

脚本文件也是纯文本文件。一般来说，脚本文件的扩展名是.js（代表JavaScript）。把脚本代码放在文件里，就要去掉<script>标签。比如，下面这些代码就是MessageScripts.js中的全部内容：

```
function showMessage() {
  alert("We interrupt this web page with a special JavaScript announcement.");
}
```

保存这个文件，然后把它放在与网页相同的文件夹里。在网页里，再定义一个脚本标签，但这次不写任何代码，而是添加一个src属性，给出刚才创建的脚本文件的路径：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>A Simple JavaScript Example</title>
  <script src="MessageScripts.js"></script>
</head>

<body>
  <p>At some point in the processing of this page, a script block
will run and show a message box.</p>
  <script>
showMessage()
  </script>
  <p>If you get here, you've already seen it.</p>
</body>
</html>
```

浏览器在遇到这个脚本标签时，会请求MessageScripts.js文件，然后就像其内容是写在网页里一样执行文件里的代码。也就是说，可以像以前一样调用showMessage()函数。

注意　在使用外部文件时，即使脚本块中不包含任何代码，也必须以</script>标签来结束它。如果不写这个结束标签，那浏览器会假设后面的一切（包括页面标记），仍然还是JavaScript代码。

也可以使用来自其他网站的JavaScript函数，只要把<script>元素的src属性指向一个完整的URL（比如http://SuperScriptSite.com/MessageScript.js）即可，只写一个文件名是不行的。这个技术对于引入其他公司的Web服务（比如Google Maps，参见13.1.5节）是非常重要的。

B.1.4　响应事件

现在，我们已经知道脚本怎样立即运行了——就是在HTML标记中放一个脚本块。但是，更常见的运行脚本的方式，其实是在页面加载之后，在用户单击按钮或把鼠标移到某个元素上的时候再运行。

为此，就需要用到JavaScript事件。JavaScript事件是在特定的事情发生时，由HTML元素发出的通知。比如，JavaScript为每个元素都赋予了一个名为onMouseOver（即“鼠标悬停”）的事件。顾名思义，这个事件会在用户把鼠标指针移动到一个元素（如段落、链接、图片、表格单元格或文本框）上面的时候发生（或用程序员的话，叫触发）。在这个操作触发onMouseOver事件后，代码文件就会执行。

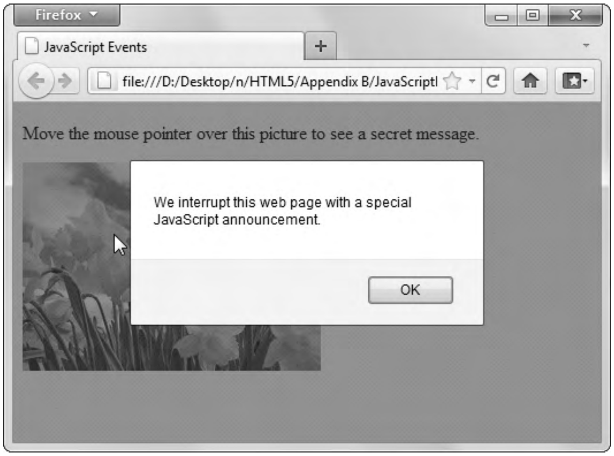
那怎么把代码与事件关联起来呢？这可是个关键的问题。方法是为相应的元素添加一个事件属性。比如，要想处理<img>元素上的onMouseOver事件，就可以这样来写：

```
<img src="sunny.jpg" alt="A sunny day" onmouseover="showMessage()">
```

注意　在JavaScript中，函数、变量和对象名都区分大小写。也就是说，showMessage和showMESSAGE不是一回事（后者在这里无效）。可是，事件属性名却不区分大小写。因为从技术角度讲，它们是HTML标记，而HTML标记允许属性是任意大小写形式。虽然如此，事件属性名全部小写（如上所示）仍然是非常常见的，一方面这样符合XHTML的书写规则，另一方面大多数程序员也都懒得去按Shift键。

这样，在鼠标移动到图片上的时候，就会触发onMouseOver事件，而浏览器则会自动调用showMessage()函数，然后显示一个消息框（图B-2）。通过事件来触发执行的函数，叫做事件处理程序。

为了有效地使用事件，应该知道JavaScript支持哪些事件。此外，还应该知道哪些元素支持哪些事件。表B-1列出了常见的事件，以及支持这些事件的元素（更完整的参考信息，请访问http://developer.mozilla.org/en/DOM/element）。



图B-2：在这个例子中，鼠标移动到图片上就会看到弹出的警告框

表B-1　常用的HTML对象事件

事　件　名	说　　明	适用元素
onClick	鼠标单击元素时触发	所有元素
onMouseOver	鼠标悬停在元素上时触发	所有元素
onMouseOut	鼠标从元素上移开时触发	所有元素
onKeyDown	按下某个键时触发	<select>、<input>、<textarea>、<a>、<button>
onKeyUp	释放某个键时触发	<select>、<input>、<textarea>、<a>、<button>
onFocus	控件接收到焦点时（也就是鼠标指针位于控件中，可以输入的时候）触发。这里所说的控件包括文本框、复选框等，请参考4.2节表4-1	<select>、<input>、<textarea>、<a>、<button>
onBlur	焦点从控件移开时触发	<select>、<input>、<textarea>、<a>、<button>
onChange	修改了控件中的值之后触发。对文本框而言，当移动到下一个控件时才会触发	<select>、<input type="text">、<textarea>
onSelect	选择输入控件中的部分文本时触发	<input type="text">、<textarea>
onError	浏览器下载图片失败时触发（通常是因为URL错误）	<img>
onLoad	浏览器下载完新页面或加载完对象（如图片）时触发	<img>、<body>
onUnload	浏览器卸载页面时触发（在浏览器地址栏中输入新URL或点击链接时发生，而且是在浏览器加载新页面前发生）	<body>

B.2　JavaScript语言基础

简明的附录难以详尽解释任何语言，即使是像JavaScript这么直观的语言也是不可能的。不过，为了帮助读者更好地理解本书中给出的示例，下面我们还是给出一些相关的基础知识。

B.2.1 变量

所有编程语言都有变量的概念，所谓变量就是一个可以在内存中保存数据的容器。JavaScript中的变量也不例外，但它是用var关键字后跟变量名声明的。下面这行代码创建了一个名为myMessage的变量：

```
var myMessage;
```

> **注意** JavaScript区分大小写，意思是myMessage与MyMessage不一样。如果你不在乎，那就会看到错误提示（当然，前提是浏览器脾气不错），或者看到页面发生错误（一般这种情况居多）。

要在变量中保存信息，要使用等号（=），等号会将其右侧的数据复制给左侧的变量。下面这个例子定义了一个变量，然后把一个文本值（文本值就是字符串）放入其中：

```
var myMessage = "Everybody loves variables";
```

然后就可以使用变量了：

```
//在消息框中显示变量中保存的文本
alert(myMessage);
```

> **注意** JavaScript是尽人皆知的松散类型的语言，即使不使用var关键字声明，照样可以使用变量。然而，不声明就使用变量却是一个非常不好的习惯，因为它会导致难以预料的错误。

B.2.2 null 值

null值是一个特殊值，程序员管它叫做空值。如果变量值是null，那就表示给定对象不存在或是空的。根据所在上下文不同，这可能表示某个功能无效。例如，Modernizr（1.6.3节）用null值判断浏览器是否支持某个HTML5功能。在脚本中也可以检测null值，比如确定还没有创建某个对象：

```
if (myObject == null) {
  //不存在myObject对象
  //现在可以创建它了
}
```

B.2.3 变量作用域

可以创建变量的地方主要有两个，一个是函数内，一个是函数外。下面的代码展示了这两种情况：

```
<script>
var outsideVariable;
```

```
function doSomething() {
  var insideVariable;
  ...
}
</script>
```

如果是在函数内部创建变量（此时叫局部变量），则该变量只在函数运行的时候存在。在此，insideVariable是一个局部变量。只要doSomething()方法一结束，这个变量就从内存中消失了。换句话说，下次再执行doSomething()方法，会重新创建insideVariable变量，与之前的那个变量毫无关系。

另一方面，如果是在函数外部创建变量（此时叫全局变量），则该变量的值会在浏览器加载页面期间始终存在。更进一步，所有函数都可以使用这个变量。对于前面的例子来说，outsideVariable是一个全局变量。

提示 经验表明，最好使用局部变量，除非确实需要在多个函数间共享变量，或者想在函数结束后仍然保留变量的值。因为如果创建的全局变量太多，管理难度会非常大，代码很容易乱套。

B.2.4 变量数据类型

在JavaScript中，变量可以保存不同类型的数据，比如文本、整数、浮点数、数组和对象。但是，无论在变量中保存什么值，都是使用同一个var关键字。换句话说，不必设置变量的数据类型。

什么意思呢？就是说，你可以给保存着文本的变量myMessage赋予一个数字值，比如：

```
myMessage = 27.3;
```

不区分变量的数据类型让JavaScript变得很灵活，因为变量可以保存的内容不受限制。与此同时，这个灵活性在不经意间也会导致JavaScript错误。比如，我们想从文本框中取得文本，然后将其放在一个变量中：

```
var = inputElement.value;
```

但是，如果不小心的话，也可能会意外地把整个文本框对象都保存到这个变量里：

```
var = inputElement;
```

这两行代码都会执行，没有任何问题。但只要几行类似的代码，就会造成将来无法恢复的错误。此时，浏览器只会停止运行其他代码，不会告诉你到底发生了什么错误。这种情况就需要JavaScript调试工具了（参见B.2.5节附注栏），它可以让你随时暂停代码执行并查看变量当前所含的数据。

B.2.5 运算

对数字值所能做的最有意义的事情，莫于对其执行运算，从而改变数据。比如，可以使用算术运算符来执行数学计算：

```
var myNumber = (10 + 5) * 2 / 5;
```

这行代码会遵循标准的数学运算法则（先计算括号里的，再计算乘法，然后计算除法，最后是加、减法）。计算结果是6。

认识JavaScript代码中的错误

为了解决问题（如前面提到的变量错误），需要掌握调试的技术。也就是说，出了问题之后，你必须能够找到错误原因，然后消灭之。可是，如何调试取决你使用的浏览器。不同的浏览器提供了不同的调试工具（或支持不同的调试扩展）。虽然它们都用于相同的目的，但使用方法却不尽相同。

好在，所有信息都可以在网上查到。下面就给出了一些链接，其中介绍了如何在不同浏览器下调试 JavaScript 错误。

- Internet Explorer。要通过 IE 调试，按 F12 键，可以看到开发人员工具窗口。至于如何使用，请参考 http://tinyurl.com/debug-ie。
- Firefox。严肃的 Firefox 开发人员都使用一个名叫 Firebug（http://getfirebug.com/javascript）的 Firefox 插件，通过它了解自己代码的运行情况。可以从 http://getfirebug.com/javascript 下载这个插件并了解更多信息 。
- Chrome。Chrome 内置了一个非常不错的调试工具。要学习使用这个调试工具，请参考谷歌的调试教程：http://tinyurl.com/c-debugger。
- Opera。Opera 提供的调试工具叫 Dragonly，可以在 www.opera.com/dragonfly 了解相关信息 。
- Safari。Safari 也有一组内置的调试工具，功能很强大，不过要查看相关使用文档却有点费劲。建议从 Safari 开发人员资料库的这篇技术文章开始：http://tinyurl.com/63om77c。

记住，解决问题与使用的浏览器和调试工具无关。只要把问题解决了，所有人就都不会再看到这个错误了。

通过运算也可以把多段文本拼接成一个长字符串。在这里，可以使用加号（+）运算符：

```
var firstName = "Sarah";
var lastName = "Smithers";
var fullName = firstName + " " + lastName;
```

现在`fullName`变量里保存着文本“Sarah Smithers.”（代码中的" "告诉JavaScript在两个名字间加上一个空格）。

在对变量进行简单的修改时，可以使用一个快捷方式。例如，以下是一个简单的加法运算：

```
var myNumber = 20;
```

```
myNumber = myNumber + 10;
//（现在myNumber值为30。）
```

但这行代码也可以简写成这样：

```
var myNumber = 20;
myNumber += 10;
//（现在myNumber值为30。）
```

像这样把运算符移到等号左侧的做法也适合其他很多运算符。下面再举几个例子：

```
var myNumber = 20;
myNumber -= 10;
//（现在myNumber值为10。）
myNumber *= 10;
//（现在myNumber值为100。）

var myText = "Hello";
var myText += " there.";
//（现在myText的值为"Hello there."）
```

另外，如果你想要给现在的变量加1或减1，还有一个更简洁的方式：

```
var myNumber = 20;
myNumber++;
//（现在myNumber值为21。）

myNumber--;
//（现在myNumber值为20。）
```

B.2.6　条件逻辑

所有条件逻辑都以一个条件开始。条件就是一个表达式，返回值为true或false。根据这个表达式的结果，可以决定是运行后面的代码，还是跳过它。

要创建条件，就要用到JavaScript的*逻辑运算符*，如表B-2所示。

表B-2　逻辑运算符

运算符	说明
==	等于
!=	不等于
===	精确等于（值和数据类型）
!==	非精确等于
!	逻辑非（对条件取反，即如果条件原来为true，现在就是false；反之亦然）
<	小于
>	大于
<=	小于等于
>=	大于等于
&&	逻辑与（如果两个表达式都为true，返回true）。如果第一个表达式为false，不会对第二个表达式求值
||	逻辑或（任意一个表达式为true，返回true）。如果第一个表达式为true，不会对第二个表达式求值

下面一个条件逻辑的简单例子：

```
myNumber < 100
```

要基于条件作决定，需要把它与if语句一起使用，如：

```
if (myNumber < 100) {
  //（如果myNumber等于20，会运行这里的代码；如果它等于147，就不会运行了。）
}
```

注意　从技术上讲，如果条件代码不超过一行，不需要使用花括号来包装，但始终使用花括号可以保证代码清晰，避免在处理多条语句时发生错误。

在测试相等性时，一定要使用两个等号。因为一个等号是设置变量的值，而不是执行比较：

```
//正确
if (myName == "Joe") {
}

//错误
if (myName = "Sarah") {
}
```

两个等号很好，但事实证明三个更好。许多JavaScript开发者更推荐用“精确等于”运算符（===）检测相等，而不仅仅是“等于”（==）。二者的不同之处在于，“等于”运算符会将两侧变量转化为同一种数据类型，而更严格的“精确等于”操作符会要求两侧变量的数据类型和值都相同。

下面这个例子演示了这种区别：

```
var myNumberAsText = "45";

// 此为真，因为“等于”运算符会将字符串"45"转换为数字45
if (myNumberAsText == 45) {
}

// 此为假，因为数据类型不匹配
if (myNumberAsText === 45) {
}
```

大多数情况下，用“等于”或“精确等于”运算符都行，但“精确等于”能防止一些不常见的类型转化。因此，JavaScript专家通常更喜欢用三个等号而不是两个。

如果想要对多个条件一个一个地求值，那么可以使用多个if块（当然啦）。但如果是想要遍历一系列条件，只找到一个匹配的表达式（忽略其他情况），则可以使用else关键字，比如：

```
if (myNumber < 100) {
  // （如果myNumber小于100，则运行这行代码。）
}
else if (myNumber < 200) {
  // （如果myNumber大于或等于100，但小于200，则运行这行代码。）
}
else {
```

```
  // （其他情况下，即myNumber大于或等于200，则运行这行代码。）
}
```

可以使用任意多个if块，而最后的else语句也是可选的。

B.2.7 循环

循环是一种基本的编程手段，可以重复地执行一个代码块。JavaScript循环的主力是for循环，它本质上是有一个包含内置计数器的循环。大多数编程语言都有自己的for循环结构。

编写for循环时，首先要设置计数器的起始值，然后再设置终止值，再设置每次循环计数器增加的值。来看一个例子：

```
for (var i = 0; i < 5; i++){
    //（这行代码执行5次。）
    alert("This is message: " + i);
}
```

循环开始是一个包含三个重要部分的括号。第一部分（即var i = 0）创建了计数器变量（i）并设置了它的初始值（0）。第二部分（i < 5）设置了终止条件。如果不是true（例如，i增加到了5），循环结束，内部的代码不再重复执行。第三部分（i++）增加计数器变量。对这个例子而言，每次循环计数器变量加1。也就是说，第一次循环i等于0，第二次循环i等于1，以此类推。最终结果是代码会运行5次，展示下列消息：

```
This is message: 0
This is message: 1
This is message: 2
This is message: 3
This is message: 4
```

B.2.8 数组

数组与for循环可谓是天作之合。数组就是一个对象，可以保存一系列值。

JavaScript数组异常灵活。与其他编程语言不同，在JavaScript中不必定义数组的项数。一开始只要使用方括号就可以创建一个空数组，比如：

```
var colorList = [];
```

然后，使用数组的push()方法可以添加元素：

```
colorList.push("blue");
colorList.push("green");
colorList.push("red");
```

或者，可以指定数组某个具体位置上的元素。如果内存中的数组还没有该位置，JavaScript就会分配一个空间：

```
colorList[3] = "magenta";
```

当然，也可以通过位置来取得数组元素的值：

```
var color = colorList[3];
```

注意 JavaScript数组使用基于零的值来存取元素。数组第一项的索引值为0，第二项的索引值为1，以此类推。

在有了保存着值的数组之后，可以使用for循环来处理其中的每个元素：

```
for (var i = 0; i < colorList.length; i++) {
  alert("Found color: " + colorList[i]);
}
```

以上代码从数组的第一项（位置0）移动到最后一项（位置以数组的length属性减1表示，length属性其实是数组中的元素个数），在消息框中显示出所有元素值。当然，这只是一个示例，你可以使用同样的方法完成更有实际意义的任务。

使用循环来处理数组是一项基本的JavaScript技术。本书示例中多次用到了这个技术，无论是你自己创建的数组，还是其他JavaScript函数返回的数组，都可以处理。

B.2.9 取得和返回数据的函数

前面我们看到了一个简单的函数showMessage()。在调用这个函数时，不需要提供任何数据，调用完成后，不会有任何信息。

并不是所有函数都这么简单。很多时候，都需要给函数传入特定的信息，或者从函数取得返回结果，在另一个操作中使用该结果。举个例子，假设我们创建一个showMessage()函数，可以通过它来显示不同的消息。为此，需要让showMessage()函数接收一个参数（也叫形参）。这个参数表示可以定制的文本，将来会在消息框中显示。

要给函数添加参数，先得给它起个名字，比如customMessage，然后把它放在函数名后的括号中，像这样：

```
function showMessage(customMessage) {
  alert(customMessage);
}
```

注意 一个函数可以接收多少个参数是没有限制的。只要把所有参数用逗号分隔开即可。

在函数内部，可以像使用变量一样使用参数。对这个例子而言，函数接收到提供给它的文本，然后在消息框中把它显示出来。

现在，再调用showMessage()函数，可以为这个函数的参数提供一个值（也就是所谓的实参）：

```
showMessage("Nobody likes an argument.");
```

通过参数可以把消息发送给函数。另外，也可以创建函数，向调用它的脚本发送回信息。发回信息的关键是位于函数末尾的return关键字，它会立即停止函数，并返回函数生成的信息。

当然，更复杂的函数既可以接收信息，又可以返回信息。比如，下面这个函数计算两个数（numberA和numberB）的乘积，然后返回计算结果：

```
function multiplyNumbers(numberA, numberB) {
  return numberA * numberB;
}
```

下面是在网页中使用这个函数的例子：

```
//传入两个数值，得到结果
var result = multiplyNumbers(3202, 23405);

//使用结果来创建一条消息
var message = "The product of 3202 and 23405 is " + result;

//显示这条消息
showMessage(message);
```

当然，你不一定要自己写函数来计算乘积（因为一行简单的JavaScript就足够了），也不一定要写函数来显示消息框（因为可以直接使用内置的alert()函数）。但这两个例子却展示了函数的基本用法，无论你要写的函数多复杂，都要像前面的例子一样使用参数和返回值。

B.2.10　对象

实际上所有的现代编程语言都包含对象的概念，它是一个相关数据与功能的包，可以在代码中与之交互。比如，在代码的眼里，网页中每个HTML元素都是一个对象。这个对象的属性可以用来读取或修改元素的内容，修改元素的样式，以及处理元素的事件。

程序员们也经常需要创建他们自己的对象。比如，在9.3.1节的画圆圈例子中，我们要创建圆圈对象；在9.4.2节的弹球例子中，我们要创建球对象。

对象简化了复杂的编程，特别是需要维护同一个数据结构的多个副本时。比如，如果需要用弹球填满页面，要创建那么多变量来保存每个球的位置和速度是件非常令人头疼的事。但如果有一种方式来声明球的模板，就可以复用那个模板创建大量的球对象，不论是一个还是一万个。

大部分语言都有创建对象模板的特定语法。通常，这些模板被称之为类。但JavaScript没有正式的类特性。这个疏忽是因为JavaScript的历史——它是作为一个简单的脚本语言问世的，而不是严格意义上的构建在线应用的工具。幸运的是，聪明的JavaScript工程师们找到了填补这个对象空白的办法。虽然这些技术在诞生之初像一种创新但古怪的巧计，但现在它们已经是标准做法了。

比如，如果要在JavaScript中定义一个对象，需要先写一个对象定义函数。这个对象定义函数就是对象模板，相当于C#、Java和Visual Basic语言中的类。下面是用来创建Person对象的对象定义函数：

```
function Person() {
  this.firstName = "Joe";
  this.lastName = "Grapta";
}
```

对象定义函数只有一个目标，它一次性定义了构成对象的所有数据。上面这个Person对象的例子中，有两个数据：名和姓（可以很容易地添加其他数据，比如生日、电子邮箱等）。this关键字很神奇——它确保了创建的每个属性都会成为对象的一部分。

只要一行以this开始，紧接其后是一个点，就可以写任何想要的属性名。所以下面这个例子是一个同等效力的Person对象，保存了同样的数据，但是用了不同的属性名：

```
function Person() {
  this.F_name = "Joe";
  this.L_name = "Grapta";
}
```

现在可以用Person()函数创建新的Person对象。具体做法不是调用函数并触发它的代码，而是用new关键字创建这个函数的一个新副本，代码如下：

```
// 创建一个新的Person对象，并保存在名为joePerson的变量中
var joePerson = new Person();
```

一旦有了一个对象，就可以通过对象定义函数中用到的属性名访问它所有的数据：

```
// 读取firstName属性.
alert("His name is " + joePerson.firstName);

// 修改firstName属性.
joePerson.FirstName = "Joseph";
```

可以优化一下这个对象定义函数，以便可以通过参数设置某些或者全部属性的值。这样就不用在创建对象之后再写几行代码来配置属性的值。这样也确保了对象以正确的状态创建出来，避免了潜在的错误。优化后的代码如下：

```
function Person(fname, lname) {
  this.firstName = fname;
  this.lastName = lname;
}
```

下面是如何用这个新的Person()函数创建两个对象：

```
var newCustomer1 = new Person("Christy", "Shanks");
var newCustomer2 = new Person("Emilio", "Taginelle");
```

9.3.1节的例子完整地呈现了创建对象的基本流程，这种技术也贯穿本书。

B.2.11 对象字面量

上一节，我们介绍了JavaScript中如何用函数创建对象，这里函数发挥了模板的作用。如果要正式定义构成一个对象的特性，使用函数是最佳方式。这样会让代码组织得更好并简化复杂的代码任务。如果需要在代码中的不同地方以不同方式使用对象，这是最佳选择。但有时我们需要一种快捷的方式来创建对象，因为只用一次。这种情况下，对象字面量很合适，因为它只需要一对花括号。

对象字面量由以下三部分构成：一个起始花括号、一个以逗号分隔的属性列表和一个结束花

括号。可以用空格和换行让代码更易读，但这不是必需的。下面是个例子：

```
var personObject = {
  firstName="Joe",
  lastName="Grapta"
};
```

每个属性，都指定了属性名和初始值。上面的代码设置personObject.firstName值为“Joe”，personObject.lastName值为“Grapta”。

13.1.4节的例子用对象字面量把信息发送给地理定位系统。只要使用了正确的属性名（getCurrentPosition()方法要求这样），对象字面量就能运行良好 。

提示 如果要了解更多有关对象字面量、对象函数的相关信息，以及JavaScript中自定义对象的任何其他相关信息，可以参阅www.javascriptkit.com/javatutors/oopjs.shtml中的详细介绍。

B.3 与页面交互

好，现在我们已经介绍了怎么在网页中使用JavaScript。可是，我们还没有做什么有意思的事儿（事实上，除了弹出一个消息框，啥还都没干呢）。在继续深入之前，有必要先了解一下JavaScript的主要角色。

首先，要知道JavaScript代码是在沙箱里运行的。什么意思？就是它的能力是受限制的。正因为如此，你的代码不能在访客计算机上执行任何有风险的操作，比如发送打印命令、访问文件、运行其他程序、格式化硬盘，等等。这样的设计确保了安全性，即使精心大意的用户也不会面临风险。

那JavaScript能做什么呢？它能做下面这些事。

- **更新页面**。脚本代码可以改变页面元素，删除页面元素，或者添加新页面元素。事实上，JavaScript可以修改当前显示的网页的任何细节，甚至能把整个文档都替换掉。
- **从服务器取得数据**。JavaScript可以向来源页面所在的服务器发送新的请求。利用这一点，再结合上述技术，就可创建出流畅地更新重要信息的网页，比如显示新闻或股票报价。
- **向服务器发送数据**。HTML已经提供了一种向服务器发送数据的方式，那就是表单（参见第4章）。不过，JavaScript则为此提供了一种巧妙的方式。你可以从表单控件取得数据，验证这些数据，然后再把它们发送给服务器。但这些操作不会刷新页面。

后两种技术都要用到XMLHttpRequest对象，12.1.1节介绍过该对象。在接下来几节中，我们主要介绍第一种技术，这是几乎所有使用JavaScript页面的根本所在。

B.3.1 操作元素

在JavaScript的眼里，你的页面可不仅仅是一个静态的HTML块。相反，页面中的每个元素都是一个对象，可以通过JavaScript代码检测和修改。

取得页面中对象的最简单方式就是通过一个唯一的名字找到它，这个唯一的名字就是ID属性。下面就是一个例子：

```
<h1 id="pageTitle">Welcome to My Page</h1>
```

像这样给元素一个唯一的ID后，就可以在代码中轻易地找到它，然后再通过JavaScript操作它。

JavaScript为寻找对象提供了简单的方法：document.getElementById()。这里的document对象表示的是整个HTML文档。这个对象始终是可以访问到的，任何时候只要你想用都可以直接引用它。与其他对象一样，document对象提供了一些有用的属性和方法。其中，getElementById()是最棒的那一个，它可以扫描整个页面，从中找到特定的元素。

注意 如果你熟悉基本的面向对象编程，对属性和方法肯定不会陌生。但如果你不熟悉的话，我可以告诉你，属性就是添加给对象的数据，而方法就是对象内置的函数。

在调用document.getElementById()方法时，要给它提供一个HTML元素的ID，以便查找。下面这个例子会在页面中查找ID为pageTitle的HTML元素：

```
var titleObject = document.getElementById("pageTitle");
```

这样，代码会找到前面看到的那个<h1>元素，把它保存在一个名为titleObject的变量中。通过把对象保存在变量中，就可以随时对它进行操作，而不必每次都重新去找它了。

那我们可以对HTML对象执行什么操作呢？从某种角度讲，可以执行的操作取决于元素的类型。例如，对超链接来说，可以改变它的URL。如果是图片，可以改变其来源地址。另外，还有一些操作是适用于所有HTML元素的，比如修改样式或者出现在开始和结束标签中间的文本内容。稍后我们就会介绍，这些技巧对于创建动态页面是非常有用的——比如，可以在访客执行某个操作时（比如点击链接时）改变页面。类似这样的交互功能，可以让访客感觉自己面对的是一个智能的、响应性的程序，而非一个呆板、简单的网页。

要修改刚才提到的<h1>元素的文本，可以像下面这样做：

```
titleObject.innerHTML = "This Page Is Dynamic";
```

以上代码用到了一个名为innerHTML的属性，该属性用于设置元素包含的内容（对这个例子来说，就是<h1>元素中的页面标题）。与其他属性一样，innerHTML只是HTML众多属性中的一个。要写出类似这样的语句，必须知道JavaScript允许你访问哪些属性。

很明显，有些属性只适用于特定的HTML元素，比如src属性用于给<img>元素加载新图片：

```
var imgObject = document.getElementById("dayImage");
dayImage.src = "cloudy.jpg";
```

另外，也可以通过style对象来修改CSS样式：

```
titleObject.style.color = "rgb(0,191,255)";
```

现代浏览器一般都会提供大量DOM属性，适用于几乎所有HTML元素。表B-3列出了其中最

有用的一些属性。

表B-3　常用的HTML对象属性

属　　性	说　　明
className	用于取得和设置class属性（参见A.2.2节）。换句话说，这个属性用于确定元素会使用什么样式（如果针对相应的类设置了样式）。当然，你得通过嵌入或链接样式表定义样式，否则只会看到简单的默认样式
innerHTML	用于读取或修改元素内部的HTML标记。这个属性极为有用，但却有两个要点要注意。首先，可以通过它来设置所有HTML内容，包括文本和标签。因此要是想给段落中的某个词加粗，可以将innerHTML设置成Hi。其次，在设置innerHTML时，会替换元素内部的所有内容，包括其他HTML元素。也就是说，如果设置了包含一些段落和图片的<div>元素的innerHTML属性，那么这些段落和图片最终都会消失，代之以新设置的内容
parentELement	保存着包含当前元素的元素对应的HTML对象。例如，当前元素是段落中的<b>元素，那么这个属性中保存的就是那个<p>元素。取得了这个父元素后，照样可以修改它
style	保存有所有CSS属性，决定着相应HTML元素的外观。技术上讲，style属性会返回完整的样式对象，如果想修改其中的样式，需要加上点（.）和要修改的样式属性的名字，比如myElement.style.fontSize。可以使用style属性来修改颜色、边框、字体，甚至定位
tagName	保存当前对象的HTML元素的标签名，不带尖括号。例如，当前对象如果表示一个<img>元素，那么这个属性会返回文本"img"

提示　HTML元素也具备一些有用的方法，其中一些可以修改属性，比如getAttribute()和setAttribute();。还有一些可以添加和删除元素，如insertChild()、appendChild()和removeChild()。要了解具体元素支持的属性和方法，请参考这里：http://developer.mozilla.org/DOM/element。

B.3.2　动态连接事件

在B.1.4节，我们看到了如何通过事件属性来连接函数。但是，通过JavaScript代码照样也可以把事件与函数关联起来。

多数情况下，你可能都会使用事件属性。但有些情况下，使用事件属性不太可能，或者不太方便。比如，当你通过代码创建了一个HTML对象并将其动态添加到页面之后，就不可能再通过事件属性来绑定处理程序。此时，页面中没有新元素的标记，也没有地方让你添加事件属性。（关于动态创建HTML对象，可以参考第8章中关于Canvas绘图的相关内容。）再比如，在需要给内置对象而非HTML元素添加事件时，也不可能使用事件属性。（可以参考第10章处理存储事件的例子。）基于以上原因，理解怎样通过代码来连接事件就非常重要了。

注意　添加事件处理程序的方式有好几种，但并不是所有浏览器都支持这些方式。本节恰好使用**事件属性**的方式，这是所有浏览器都支持的。但如果你在使用jQuery等JavaScript工具包，那么也可能会用到它的另一套事件绑定机制，这套机制不仅所有浏览器都支持，而且还具备其他一些功能。

好在添加事件非常简单，只要像添加事件属性（property）一样，在代码中设置事件属性（attribute）即可。例如，假设你的页面中有如下<img>元素：

```
<img id="dayImage" src="sunny.jpg" alt="The weather">
```

如果你想让用户单击这张图片时调用swapImage()函数，那么可以这样：

```
var imgObject = document.getElementById("dayImage");
imgObject.onclick = swapImage;
```

不过，千万不要犯这样的错误：

```
imgObject.onclick = swapImage();
```

这样会运行swapImage()函数，得到返回的结果（如果该函数有返回值的话），然后用该函数返回的结果设置事件处理程序。这肯定不是你想要的。

要理解用户单击<img>元素时会发生什么，需要看一看swapImage()函数的代码。这个函数会取得<img>元素，然后修改它的src属性以指向新图片（参见图B-3）：

```
//记录下从白天到晚上是否更换了图片
var dayTime = true;

//这个函数会在onClick事件发生时执行
function swapImage() {
  var imgObject = document.getElementById("dayImage");

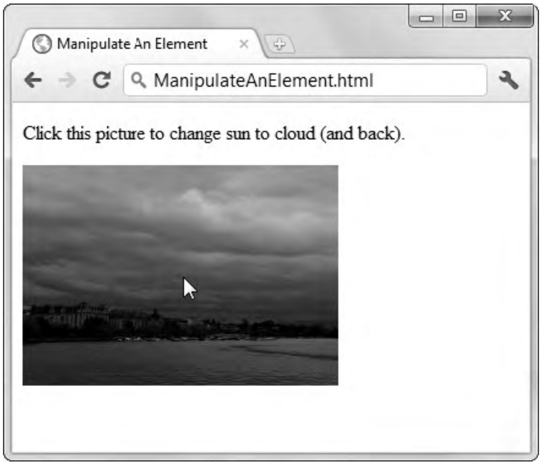
  //白天到晚上，或晚上到白天都要更换图片
  if (dayTime == true) {
    dayTime = false;
    imgObject.src = "cloudy.jpg";
  }
  else {
    dayTime = true;
    imgObject.src = "sunny.jpg";
  }
}
```

有时候，事件会给事件处理程序传递有价值的信息。要使用这些信息，需要为事件处理程序添加一个参数。按照惯例，这个参数通常命名为event或干脆命名为e：

```
function swapImage(e) {
  ...
}
```

这个事件对象有哪些属性，取决于具体的事件。例如，对于onMouseMove事件，其事件对象会提供鼠标的当前坐标值（利用这个坐标值可以创建类似8.2节的绘画程序）。

还有一点值得注意。如果代码连接到了事件，那你必须把整个事件名小写。这一点与在HTML中使用事件属性是不一样的。与JavaScript不同，HTML不在乎大小写问题。



图B-3：单击图片，页面就会触发事件，该事件会调用一个函数，该函数会加载新图片

注意 本书的事件采用了容易看懂的驼峰式大小写方式，即每个单词的首字母大写（如onLoad或onMouseOver）。不过，在脚本代码中，所有事件都采用了小写形式（如onload和onmouseover），因为JavaScript在乎。

B.3.3 嵌入事件

为了让前面的例子运行起来，必须在代码中的某个位置定义swapImage()函数。有时候，可能需要跳过这一步，直接在想要添加事件处理程序时定义函数。这种技术就叫做*嵌入函数*。

以下是一个为onClick事件添加嵌入函数的例子：

```
var imgObject = document.getElementById("dayImage");
imgObject.onclick = function() {
  //原来swapImage()函数的代码
  //现在放在这里了
  if (dayTime == true) {
    dayTime = false;
    imgObject.src = "cloudy.jpg";
  }
  else {
    dayTime == true;
    imgObject.src = "sunny.jpg";
  }
};
```

这种快捷写法比使用独立的命名函数少见一点，但这种方式仍然是非常方便的，本书示例偶尔会用到这种写法。

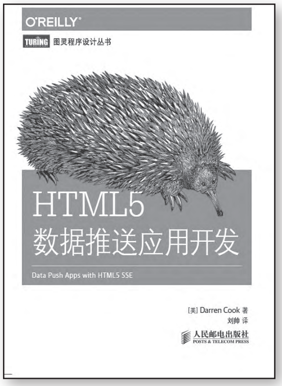
注意 嵌入函数在处理**异步**任务时会很有用，异步任务就是在后台执行的任务。当异步任务完成时，浏览器会触发事件，通知执行相应的代码。有时候，处理这种情况的最清楚的方式，就是在任务**开始**的位置放置处理任务**完成**的代码。（7.1.1节有一个例子，展示了异步加载和处理图片，可以作为参考。）

最后，有一个嵌入函数的例子在本书很多示例中都用到了。那就是window对象的onLoad事件处理程序，该函数会在页面加载、显示、准备就绪时执行。此时，正好是代码披挂上阵的时候。如果在此之前运行代码，可能会遇到问题，比如对应某些元素的对象还没有创建完成：

```
<script>
window.onload = function() {
  alert("The page has just finished loading.);
}
</script>
```

使用这种方式，可以不必担心脚本块的位置。你照样可以把初始化代码放在<head>区块中，与其他JavaScript函数放在一起。

更多推荐



HTML5 数据推送应用开发
作者：Darren Cook
书号：978-7-115-37059-4
定价：49.00 元



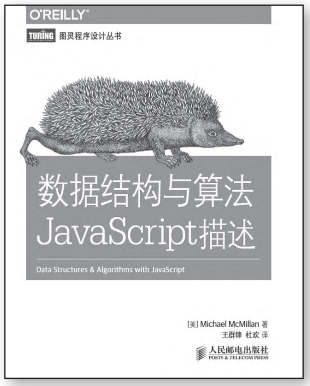
HTML5 与 CSS3 实例教程（第 2 版）
作者：Brian P. Hogan
书号：978-7-115-36340-4
定价：49.00 元



HTML5 与 CSS3 基础教程（第 8 版）
作者：Elizabeth Castro Bruce Hyslop
书号：978-7-115-35065-7
定价：69.00 元



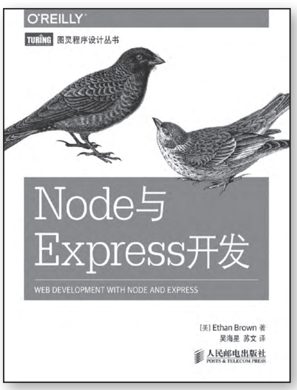
JavaScript 快速全栈开发
作者：Azat Mardanov
书号：978-7-115-37609-1
定价：49.00 元



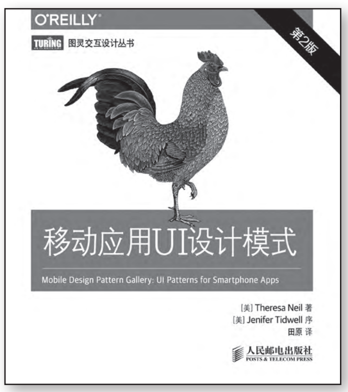
数据结构与算法 JavaScript 描述
作者：Michael McMillan
书号：978-7-115-36339-8
定价：49.00 元



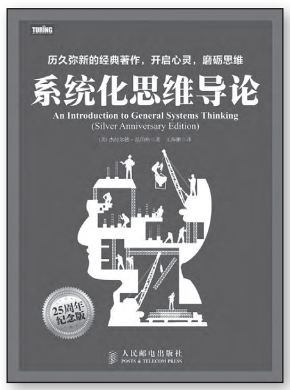
JavaScript 编程实战
作者：Jon Raasch
书号：978-7-115-34548-6
定价：59.00 元



Node 与 Express 开发
作者：Ethan Brown
书号：978-7-115-38033-3
定价：69.00 元



移动应用 UI 设计模式（第 2 版）
作者：Theresa Neil
书号：978-7-115-37790-6
定价：79.00 元



系统化思维导论
作者：杰拉尔德·温伯格
书号：978-7-115-37804-0
定价：59.00 元

HTML5 / WEB DEVELOPMENT



答案尽在本书！

HTML5不仅仅是一种标记语言，它还指代多种独立的Web标准。你的运气不错：这本书涵盖了关于HTML5的一切！本书通过循序渐进的教程和真实的示例，向你介绍如何构建Web应用，包括视频控件、动态图形、地理定位、离线功能和针对移动设备的响应式Web设计等。

the missing manual®

The book that should have been in the box®

本书主要内容如下：

- 不依赖插件添加音频和视频，构建适用于所有浏览器的播放页面；
- 用Canvas创建吸引人的视觉效果，绘制图形、图像、文本，播放动画，运行交互游戏；
- 用CSS3将页面变活泼，比如添加新奇的字体，利用变换和动画添加吸引人的效果；
- 设计更出色的Web表单，利用HTML5新增的表单元素更加高效地收集访客信息；
- 一次开发，多平台运行，实现响应式设计，创建适配桌面计算机、平板电脑和智能手机的网站；
- 让Web应用拥有桌面应用的特性，通过开发自给自足的离线应用，无论用户能否上网，都可以在本地保存用户信息。



Matthew MacDonald

著名科技作家、程序员，撰写过十几部技术专著，包括《ASP.NET 4高级程序设计》《精通ASP.NET 4.5（第5版）》，以及Missing Manual系列图书之Creating a Website、WordPress、Access 2013、Office 2013、Excel 2013、Your Brain和Your Body等。他的个人网站是prosetech.com。

图灵社区：iTuring.cn
热线：(010)51095186转600

O'REILLY®
missingmanuals.com

分类建议 计算机/Web开发/HTML5

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

ISBN 978-7-115-32050-6

9 787115 320506 >

ISBN 978-7-115-32050-6
定价：89.00元

看完了

如果您对本书内容有疑问，可发邮件至contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这里可以找到我们：

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks